ויליאם פרג'ון

פסקל
מהצעד הראשון

Turbo Pascal
הגירסה האחרונה



הוצאת הוד־עמי
לספרי מחשבים

פסקל
מהצעד הראשון

מהדורה שנייה

עורך ראשי: **יצחק עמיהוד**

עריכה: **שרה עמיהוד**

ייעוץ מקצועי: **יהודה כץ**

מוקדש

לאמי מרים פרג׳ון

לשם שטף הקריאה כתוב ספר זה בלשון זכר בלבד. ספר זה מיועד לגברים ונשים כאחד ואין בכוונתנו להפלות או לפגוע בציבור המשתמשים/ות.

- **טלפון: 09-9564716**
- **פקס: 09-9571582**
- **דואר אלקטרוני: info@hod-ami.co.il**
- **אתר באינטרנט: www.hod-ami.co.il**

פסקל
מהצעד הראשון

מהדורה שנייה

ויליאם פרג'ון



הוצאת הוד-עמי
לספרי מחשבים

Pascal - From the First Step on

William Farjon

Editor**: I. Amihud**



הוצאת הוד-עמי
לספרי מחשבים בע"מ
ת.ד. 6108, הרצליה 46160
טל': 09-9564716 פקס: 09-9571582

בקרו אותנו באינטרנט: **www.hod-ami.co.il**
כתבו לנו: **info@hod-ami.co.il**

הודפס בישראל
סיון תשנ"ה, יוני 1995
עדכון: 1996, 1999
מהדורה שנייה 2000

ISBN 965-361-062-7 מסת"ב

תוכן עניינים מקוצר

תוכן העניינים

פרק 6: פרוצדורות ופונקציות 83

פרק 7: מחרוזות 105

פרק 8: מערכים 119

פרק 1

מבוא לשפת פסקל

הקדמה

אודות שפת פסקל

שפת פסקל תוכננה במקור ללימוד כללי תחביר, אופן תכנות, פיתוח חשיבה לוגית, וכבסיס ללימוד שפות אחרות. כשפת לימוד, תחביר השפה ואופן השימוש בה פשוט ביותר. עם השנים התפתחו גרסאות שונות לשפה. כל חברת תוכנה שעסקה בנושא הרחיבה את תחומי השפה וכך יצרה מעין שפה חדשה. רק הבסיס של השפה משותף לכל הגרסאות. כיום נותרו רק גרסאות בודדות לשפת פסקל המקורית. אחת מהן היא גרסת **טורבו פסקל** (**Turbo Pascal**) של **חברת בורלנד** (**Borland**). זו אחת הגרסאות הנפוצות ואותה נלמד בספר זה. לכן, כל התייחסות לשפת פסקל מעתה ואילך מכוונות למהדר - **טורבו פסקל**.

אופי ומבנה השפה

שפת פסקל פשוטה להבנה ושימוש, אך יחד עם זאת היא גם מאפשרת ליצור מערכות תוכנה מורכבות. פסקל, כמו כל שפת תכנות, מחייבת כתיבת פקודות במבנה קבוע ומוגדר, וזאת נלמד בספר זה.

מרכיבי שפת פסקל

בפרק זה נעסוק במרכיבים הבסיסיים של שפת פסקל:

א) משתנים וקבועים.

ב) מבנה כללי של תוכנית.

ג) פקודות קלט/פלט פשוטות.

מילים שמורות

מילים שמורות מקובלות בכל שפת תכנות. הן שייכות לשפה עצמה, ולא ניתן להשתמש בהן, אלא רק כחלק מפקודה. לפניך רשימת המילים השמורות. המילים המסומנות בכוכבית מתייחסות לטורבו-פסקל בלבד, ואינן שייכות לשפת פסקל הסטנדרטית:

***ABSOLUTE**	**NOT**	***IMPLEMENTATION**	**PACKED**	**TO**
AND	**RECORD**	***INTERFACE**	**PROCEDURE**	**TYPE**
ARRAY	**THEN**	***INTERRUPT**	**PROGRAM**	***UNIT**
BEGIN	**VAR**	**LABEL**	**RECORD**	**UNTIL**
CASE	**ARRAY**	**MOD**	**REPEAT**	**USES**
CONST	**DOWNTO**	**NIL**	**SET**	**VAR**
DIV	**ELSE**	**NOT**	***SHL**	**WHILE**
DO	**END**	**OF**	**SHR**	**WITH**
FOR	**IF**	**OR**	**STRING**	**XoR**
***INLINE**	**IN**		**THEN**	

משתנים וקבועים

משתנה (**variable**) כשמו כן הוא - משתנה. תפקידו לאגור ערכים שונים ולהשתמש בהם בשלבים אחרים של התוכנית. כדי להבין את משמעותו, נתבונן לדוגמה בסיר המכיל מים. ניתן להוסיף לו מים, להפחית ממנו מים ואחר-כך להשתמש במים שנשארו. כך גם המשתנה, הוא אוגר בתוכו נתון מסוים ובשלב אחר של התוכנית משתמשים בנתון השמור בו.

שפת פסקל כוללת סוגים אחדים של משתנים בסיסיים (השמות בסוגריים מציינים את השמות הלועזיים שלהם):

משתנה שלם (Integer): משתנה זה אוגר מספרים שלמים בלבד. כל ערך ששמים בתוכו, בין אם חיובי או שלילי, חייב להיות שלם.

משתנה ממשי (Real): משתנה זה מכיל ערכים מספריים, הכוללים שבר עשרוני. משתנה ממשי מכיל מספרים ממשיים, שהם ערכים שכוללים שבר. לדוגמה: 234 או 1 או 9.194231-.

משתנה תו (Char): משתנה זה שונה מקודמיו, כי התוכן שלו אינו ערך מספרי שלם או שבר, אלא תו בודד. לדוגמה: "א", "Q" "ז" "6". מלבד העובדה שהוא מכיל תווים בלבד, השוני העיקרי ממשתנים אחרים הוא בכך שלא ניתן לבצע עמו חישובים. ניתן אך ורק לשמור בו נתון או לקרוא ממנו נתון.

משתנה לוגי/בוליאני (Boolean): משתנה לוגי מייצג שני מצבים: "כן" ו"לא", או Yes/No, כשמדובר באנגלית. המשתנה הלוגי (משתנה זה נקרא בדרך כלל בוליאני, על פי שמו הלועזי) דומה למדי למשתנה השלם. ניתן לבצע עמו חישובים לוגיים, כלומר, כפל לוגי וחיבור לוגי, וכל זאת בתנאי ששני המשתנים או הערכים שפועלים אתם הם משתנים או ערכים בוליאניים. לא ניתן לבצע פעולות במשתנה זה עם משתנים אחרים, כגון משתנים שלמים או ממשיים. במשתנים מסוג זה נדון בפרק 3, מכיון שלגביהם קיים תחום מיוחד הנקרא "אלגברה בוליאנית".

בשפת פסקל, כמו ברוב השפות האחרות, ניתן להגדיר **קבועים** (**Constants**) מסוגים שונים. למעשה, כל סוג שניתן להגדיר כמשתנה (לדוגמה, שלמים, ממשיים, תווים וכו') ניתן להגדיר גם כקבוע. כבר לפי השם ניתן להבחין בהבדל. כאשר מגדירים משתנה, ניתן להשתמש בו ולשנות את תוכנו במהלך התוכנית. כאשר מגדירים קבוע, ניתן אך ורק לקרוא ממנו ללא כל יכולת לשנותו בזמן ההרצה. הערך ההתחלתי שיוגדר עבור הקבוע יישאר "קבוע" למשך כל זמן הרצת התוכנית.

מבנה התוכנית בשפת פסקל

עד כה למדנו מהם משתנים וקבועים, ולמדנו איזה סוגים בסיסיים קיימים. כעת עלינו לדעת כיצד מגדירים אותם בתוכנית. כדי להתקדם לשלב הזה עלינו לדעת תחילה מהו המבנה הכללי של תוכנית. שפת פסקל היא שפה מבנית, ולכן לכל מרכיב בתוכנית יש מיקום מסוים שנקבע מראש. למשל, אי אפשר לערבב הגדרת משתנים עם פקודות התוכנית.

מבנה תוכנית כולל שלושה חלקים עיקריים:

- ראש התוכנית
- הגדרת משתנים
- החלק הביצועי של התוכנית

ראש התוכנית

חלק זה הוא למעשה הגדרת שם התוכנית. לרוב הדבר מסתכם בשורה אחת, או שתיים לכל היותר. בשלב זה נאמר רק, כי חלק זה מכיל שורה אחת הנראית כך:

```
program Name;
```

המילה **program** מבהירה למחשב כי תפקיד שורה זו לציין את שם התוכנית. כלומר, **Name** הוא שם התוכנית. לדוגמה, נכתוב:

```
program Copy;
```

שורה זו מגדירה למחשב כי שם התוכנית הוא Copy. ניתן לציין כל שם שרוצים, ובלבד שהוא נכתב במבנה נכון ולפי הכללים:

- שם חייב להתחיל באות אנגלית בלבד.
- אסור לציין יותר ממילה אחת בשם התוכנית (לדוגמה, שם כגון "Copy File" הוא שגוי, מכיון שהוא מכיל יותר ממילה אחת).
- כל התווים חייבים להיות אותיות אנגליות או ספרות, או קו תחתי (_).
- אסור שהשם יהיה מילה שמורה.

הגדרות אלו תקפות גם לשאר המקרים בהם אנו משתמשים בשמות. מקרים אחרים יכולים להיות שמות של משתנים, קבועים וגם פקודות בתוך התוכנית. כולם חייבים להיות בעלי שמות תקניים על פי התבנית שהגדרנו.

למעשה, שורה זו אינה משנה מאומה בתוכנית, אשר יכולה להתבצע ללא תלות, אם שורה זו קיימת או לא.



הערה חשובה: *לאחר כל פקודה או הגדרה בשפת פסקל חייב להיות התו ";" (נקודה-פסיק). רק במקרים בודדים אין צורך להשתמש בתו זה, כפי שנלמד בהמשך.*

הגדרת משתנים

בחלק זה מוגדרים כל המשתנים והערכים הקבועים שנשתמש בהם בהמשך התוכנית. כדי שהמחשב יוכל להבחין מהו משתנה ומהו קבוע, יש לחלק את ההגדרה לשתי קבוצות שונות. תחילה, נדגים כיצד מגדירים משתנה פשוט:

```
var
   Name:                 Type;
```

המילה var מגדירה למחשב כי מנקודה זו של התוכנית והלאה, כל עוד לא ציינו אחרת, אנו מגדירים משתנים. כדי לפעול עם משתנים אנו צריכים לקבוע להם שמות, כי המחשב אינו יכול לנחש בעצמו באילו משתנים אנו רוצים להשתמש. לצורך זה Name מציין את שם המשתנה, אשר חייב להיות שם תקני. Type הוא סוג המשתנה, לדוגמה משתנה שלם, ממשי, תו וכו'.

שימו לב: *אנו כותבים את ההגדרות וההוראות בהזחה ועם טבלר (Tab), מכיון שכך נכון להציג את התוכנית. בדרך זו היא תהיה קריאה וברורה יותר, ואין לכך חשיבות מבחינת השפה. השתדל לנהוג לפי הדוגמאות שבהמשך.*

הפיסקה הבאה מגדירה משתנה "X" מסוג **שלם** (**Integer**):

```
var
   X:        Integer;
```

שימו לב: *אחרי המילה "var" אין לכתוב את התו "**;**".*

דוגמאות נוספות להגדרת משתנים:

```
var
   R:        Real;
   Ch:       Char;
   Check:    Boolean;
   I:        Integer;
```

כעת נבחן כיצד מגדירים קבועים. ערכים קבועים קלים יותר להגדרה מכיון שהמהדר של פסקל אינו דורש מאתנו להגדיר לו את סוג המשתנה, הוא מחליט בעצמו על פי הערך שאנו מציבים בו. הגדרת פסקת הקבועים נראית כך:

```
const
   Name = constant_value
```

המילה "**constant**" מגדירה למחשב כי מנקודה זו והלאה, וכל עוד לא ציינו אחרת, כתובות הגדרות של קבועים. לדוגמה, הפיסקה הבאה מגדירה קבוע בשם Pi ומציבה בו את הערך 3.14:

```
const
   Pi = 3.14;
```

דוגמאות נוספות להצבות קבועות:

```
const
   I = 11;          משתנה שלם
   Ch = 'A';        משתנה תו
   R = 11.1;        משתנה ממשי
```

לאחר שלמדנו את הדרך להגדרת משתנים וקבועים, אנו יכולים לשאול שאלה מעשית: איזה ערך יימצא במשתנה שהוגדר, אך עדיין לא נעשה בו שימוש? נתון כזה מכיל בדרך כלל את הערך אפס, אך אין זו עובדה קבועה **ואין להסתמך על כך**. אם כן, מה עלינו לעשות כדי שנוכל להגדיר משתנה עם ערך שיוצב בו מראש לפני השימוש הראשוני בו? לשם כך ניתן לשלב שתי תכונות. האחת היא תבנית הגדרת משתנה, והשנייה - הצבת ערך קבוע. כיצד הדבר מתבצע?

הפיסקה הבאה מגדירה משתנה שלם עם ערך התחלתי:

```
const
   I:       Integer = 21;
```

פיסקה זו הגדירה **משתנה**, ובו ערך התחלתי 21. כעת נוכל להיות בטוחים כי משתנה זה מכיל את הערך 21 טרם שימושו במהלך התוכנית.

החלק הביצועי של התוכנית

החלק המרכזי של התוכנית הוא החלק אשר בו אנו מכתיבים למחשב מה עליו לבצע. בתחילת קטע זה תמיד תהיה המילה השמורה **Begin** (**התחלה**), ובסופו המילה השמורה **.End** (**סוף**).

שימו לב: *לאחר המילה End שבסוף התוכנית כותבים* ***נקודה.***

הדוגמה הבאה הינה שלד שלם של תוכנית:

```
program Dugma;
const          הגדרת קבועים ומשתנים מתוחלים מראש
var            הגדרת משתנים
begin
   ...
   ...         מהלך התוכנית
end.
```

פקודות קלט/פלט (Readln\Writeln)

כל אשר למדנו עד כה יכול לעזור בבניית משוואות פשוטות, תרגול משתנים והבנת הלוגיקה של המחשב. דבר אחד חשוב עדיין אין אנו יודעים: קלט ופלט - כיצד להזין נתונים ולהפיק תוצאות. בני אדם לא היו יכולים לתקשר אם לא היו להם חושים שעוזרים להם בכך. כך גם במקרה של המחשב, המחשב אינו יכול להדפיס את ההודעה ללא פקודות פלט, או לקבל נתונים ללא פקודות קלט.

בשפת פסקל יש שתי פקודות בסיסיות המאפשרות לקלוט ולהדפיס נתונים. כאשר אנו רוצים לקלוט נתון, נשתמש בפקודה **Readln** (**קרא**), וכאשר נרצה להדפיס נתון או הודעה נשתמש בפקודה **Writeln** (**הדפס**).

כאן עלינו לעצור וללמוד תחילה מהו מבנה הפקודות בשפת פסקל, וכיצד משתמשים בהן. באופן כללי, ניתן לתאר מבנה של פקודה באופן הבא:

```
command_name(param1, param2, ...);
```

command_name הוא שם הפקודה (לדוגמה Readln, Writeln, או פקודות אחרות שנלמד בהמשך).

param1, param2 וכו' הם הנתונים שאנו מעבירים לפקודה.

מכיוָן שהמחשב אינו יכול "לנחש" אילו נתונים אנו רוצים להדפיס, לקלוט או לעבד, אנו חייבים לציין זאת באופן מפורש. קיימות פקודות שדורשות יותר מנתון **אחד** ולכן, כשקיים מצב כזה, אנו מציינים את הנתונים משמאל לימין (השמאלי ביותר הוא הראשון, והימני ביותר הוא האחרון) ומפרידים ביניהם בפסיק (,).

כעת ניגש להגדרת **הפקודה Readln**. התבנית שנשתמש בה לפקודה זו :

```
Readln (var_name);
```

var_name מציין את שם המשתנה שאליו אנו רוצים לקלוט נתון. הדוגמה הבאה קולטת מספר שלם לתוך המשתנה I, וכופלת אותו בשניים :

```
program Input;
var
   I:          Integer;
begin
   Readln (I);
   I := I * 2;
end.
```

נתבונן כעת בתוכנית וננסה לחשוב מה חסר בה. במבט ראשון היא נראית כשורה, והמחשב אף יאשר את הרצתה, כך שמבחינה טכנית הכל בסדר. נבחין כי אין אנו מבצעים כל פעולה ממשית לאחר קריאת נתון שלם. לפקודת החישוב אין כל חשיבות אם אין מאחסנים את התוצאה, או מדפיסים אותה. לצורך הדפסת נתונים קיימת פקודה מיוחדת שכבר הספקנו להכיר. לפקודה זו קוראים **Writeln**. תבנית פקודה זו קצת יותר מורכבת מזו של הפקודה Readln מכיוָן שבעזרת ההדפסה ניתן להדפיס יותר מנתון אחד, ואף לבצע חישובים מסוימים. תחילה נלמד את אופן כתיבת הפקודה :

```
Writeln (param1, param2, param3, ... );
```

השאלה הראשונה שכבר עתה ניתן לשאול היא מהם הנתונים הדרושים לפקודה Writeln? התשובה לכך פשוטה : הפקודה Writeln היא פקודה מיוחדת, המקבלת כל מספר של משתנים או צירוף שלהם. בהמשך נלמד שיש פקודות בעלות מספר קבוע של פרמטרים ועלינו לציין את כולם. כלל זה אינו חל על פקודה זו. נתבונן בתוכנית הבאה :

```
program Dugma;
var
   I:          Integer;

begin
   Readln (I);
   I := I * 2;
   Writeln ('The result is: ', I);
end.
```

אנו רואים כי תוכנית זו זהה לתוכנית הקודמת, בשינוי של שורה אחת. בשורה שלפני האחרונה קיימת תוספת : הדפסת התוצאה **בשילוב** של הודעה מתאימה.

נבחן שורה נוספת המדגימה את השימוש בפקודה Writeln :

```
Writeln ('Original value is: ', I, ' the result is: ', I*2);
```

שימו לב: *כשאנו רוצים להדפיס שתי הודעות ושני ערכים שונים, הערך הראשון מיוצג על ידי המשתנה I, והערך השני הוא תוצאת חישוב. נשים לב לכך שאפשר לכתוב "נוסחה" במקום שם של משתנה. תחילה תחושב הנוסחה, ולאחר מכן תודפס התוצאה.*

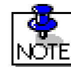
נציין גם, כי את ההודעות תחמנו על ידי שני גרשים ('), אחד מכל צד. כעת אנו יכולים אף לשפר את התוכנית, ולכתוב אותה כך :

```
1)  program Dogma;
2)  var
3)    I:              Integer;
4)
5)  begin
6)    Writeln ('Type a number: ');
7)    Readln (I);
8)    Writeln ('Result is: ', I*2);
9)  end.
```

הערה: *המספרים שמשמאל בכל שורה נכתבו לצורך ההסבר בלבד והם **אינם** חלק של התוכנית.*

ננסה להבין מה תוכנית זו עושה :

בשורה 3 התוכנית מגדירה משתנה "I" מסוג שלם. בשורה 6 המחשב ידפיס הודעה שתבקש מאתנו להקליד מספר כלשהו. בשורה 7 התוכנית תקלוט את המספר אל המשתנה "I", ובשורה 8 היא תדפיס הודעה בצירוף תוצאת החישוב.



נזכור *לגבי הפקודה Writeln שני דברים חשובים :*

א) פקודה זו יכולה לקבל מספר כלשהו של פרמטרים (נתונים). כלומר, אין אנו חייבים לציין לה מספר מסוים של פרמטרים בכל הרצה.

ב) לפקודה יש תבנית גמישה. נוכל להדפיס הודעות בשילוב משתנים או ערכים כפי שנרצה.

עתה, משהכרנו את שתי הפקודות הללו, נוכל להרחיב את ההסבר בנושא קלט/פלט. לימוד פקודות אלו בצורה בסיסית יאפשר לנו להבין את משמעותן ואופן השימוש בהן. נפנה כעת למבנה המורחב יותר של הפקודה Readln.

ננסה לבחון מצב שבו אנו רוצים לקרוא כקלט שלושה ערכים ממשיים, כמו לדוגמה, מיקום במרחב (X, Y, Z). בשיטה הקונבנציונלית שלמדנו עתה נוכל לקרוא את שלושת הנתונים הללו בשלוש קריאות נפרדות, קריאה בשלוש פקודות Readln. לחילופין, מציעה לנו שפת פסקל דרך נוחה יותר לביצוע מטלה זו: קריאת שלושת הנתונים הללו תוך שימוש בפקודה Readln אחת. כיצד נעשה זאת?

בדומה לפקודה Writeln, שבעזרתה ניתן להדפיס כמה מבנים של נתונים תוך הפרדתם על ידי התו "," (פסיק), כך גם ניתן לקרוא נתונים אחדים בעזרת פקודת Readln אחת. נתבונן בקטע התוכנית הבא:

```
program Dogma;
var
   X, Y, Z:   Real;

begin
   Readln (X, Y, Z);
end.
```

בתוכנית זו הגדרנו שלושה משתנים ממשיים (X, Y, Z) וקראנו לתוכם שלושה ערכים, וכל זאת בעזרת פקודת Readln יחידה. כעת נתבונן גם על הפן השני של פקודה זו - מבנה הקלט. כאשר אנו מורים למחשב לבצע פקודה זו, הוא מצפה לקבל שלושה ערכים ממשיים, שמופרדים ביניהם ברווח. לדוגמה, לאחר שנריץ את התוכנית, המחשב ימתין עד שנכניס שורה כגון זו:

```
14.3  15.2  -3.5
```

הערך 14.3 יוצב במשתנה X, הערך 15.2 יוצב במשתנה Y, והערך -3.5 יוצב במשתנה Z.

איך כותבים תוכנית קריאה וברורה



שימו לב *שוב לכך שאנו כותבים את שורות התוכנית* ***בהזחה*** *(הזזה -* ***indentation****) מסוימת לימין. ההזחה* ***אינה*** *דרושה לשפת פסקל, אלא לנו, הכותבים וקוראים את התוכנית. תוכנית הכתובה בצורה זו יותר ברורה, מכיון שהיא מבליטה את קטעי התוכנית ואת השייכות של פקודות לפקודות אחרות. אנו ממליצים לנהוג לפי הדוגמאות שבספר.*

פרק 2

אופרטורים חישוביים

מהם אופרטורים?

המילה **אופרטור** (**operator**) נובעת מהמילה הלועזית operation, שפירושה "פעולה". ואמנם, המילה "אופרטור" מייצגת פעולה כלשהי. בפרק זה נדון באופרטורים המבצעים פעולות חישוב. בין השאר נלמד כיצד מבצעים את **ארבע פעולות החשבון הבסיסיות** (**חיבור**, **חיסור**, **כפל וחילוק**), **חישוב מנה ושארית**, **וכיצד מפעילים פונקציות מתמטיות בסיסיות** לחישוב ערכים מוחלטים, ערכים ריבועיים, שורשים ועוד.

פעולות חשבון בסיסיות

כאשר עוסקים בחישובים מתמטיים בשפת פסקל יש לזכור את הכלל הבסיסי:
כתיבת תהליך חישוב במחשב תואם בתבניתו לכתיבת **משוואה**.

לדוגמה, את המשוואה y = x + 1 נכתוב כך:

```
program Targil;
var
    x, y:       Real;

begin
    y := x + 1;
end.
```

בתוכנית זו הגדרנו שני משתנים ממשיים ויישמנו את המשוואה y = x + 1. כפי שניתן להבחין, תבנית כתיבת המשוואה בשפת פסקל דומה לחלוטין לכתיבת משוואה מתמטית. ההבדל היחיד שניתן להבחין בו הוא, שבנוסף לסימן השוויון אנו מוסיפים את התו נקודתיים (:) ומקבלים "=:" המייצג פעולת השמה למשתנה ולא שוויון.

ומה לגבי שאר פעולות החשבון? את פעולת ה**חיסור** נבצע, כמובן, עם התו "-", ואת פעולת ה**כפל** נבצע עם התו "*" (כוכבית). מכיון שקיימים שני סוגים של משתנים חשבוניים: השלמים (שמכילים ערכים שלמים בלבד ללא שברים) והממשיים (הכוללים ערכים בצירוף שברים עשרוניים), עלינו להשתמש בשתי שיטות נפרדות כדי לציין פעולת חילוק. כאשר עוסקים בערכים ממשיים, מבצעים את פעולות ה**חילוק** בעזרת התו "/" (לוכסן), וכאשר מבצעים פעולת חילוק בערכים שלמים משתמשים במילה השמורה "**div**" (קיצור המילה הלועזית **division** - חילוק). לדוגמה, אם נתונים שני **ערכים ממשיים** (X ו-Y) שעלינו לחשב את המנה שלהם (Z), נכתוב:

```
Z := X / Y;
```

בשורה זו מחלקים את X ב-Y (חילוק ערכים ממשיים) ומציבים תוצאה ממשית במשתנה Z. כאשר נתונים שני משתנים (A ,B) מסוג **שלם**, ונרצה לחשב את המנה השלמה (C), נכתוב:

```
C := A div B;
```

בשורה זו ביצענו פעולת חילוק בין שני מספרים מסוג שלם B,A(integer) והתוצאה במשתנה C אף הוא מסוג שלם.

בנוסף לארבעת האופרטורים הבסיסיים (חיבור, חיסור, כפל וחילוק) קיים אופרטור נוסף, אשר ניתן להשתמש בו רק עם ערכי שלמים. זהו **אופרטור השארית - mod**, הדומה לכל אופרטור חשבוני אחר. על ידי אופרטור זה אנו יכולים לחשב את השארית של חלוקת שני מספרים. לדוגמה, כדי לקבל את השארית מחלוקת המספר 9 ב-4 נרשום:

```
A:= 9 mod 4;
```

בשורה הבאה חישבנו את השארית (מספר שלם) של חלוקת B ב-4.

```
x:= B mod 4;
```

נבחן עתה את התוכנית הבאה, בדוגמה זו נקבל ב-A את הערך 1 (בדקו למה):

```
program Targil;
var
  A, B, C, D:    Integer;

begin
  Writeln ('Type first number:');
  Readln (A);
  Writeln ('Type second number:');
  Readln (B);
```

```
10)    C := A div B;
11)    D := A mod B;
12)    Writeln ('Mana = ', C);
13)    Writeln ('Shearit = ', D);
14) end.
```

בשורה 3 אנו מגדירים ארבעה משתנים שלמים שבהם נאחסן את הקלט ואת תוצאות החישובים שמיועדים לפלט. בשורה 6 ידפיס המחשב הודעה בנוסח: ״הכנס את הערך הראשון״, ולאחר מכן יקלוט ערך זה לתוך המשתנה A (שורה 7). בשורה 8 נעשה דבר דומה, המחשב ידפיס הודעה בנוסח: ״הכנס את הערך השני״ ויקלוט ערך זה לתוך המשתנה B (שורה 9). בשורה 10 תחושב המנה של חלוקת שני הערכים הללו והתוצאה תוכנס למשתנה C. בשורה 11 תחושב השארית של חלוקת שני הערכים הללו והיא תיכתב במשתנה D. כעת, נותר להורות למחשב להדפיס את תוצאות החישובים. לכן, בשורה 12 הורינו להדפיס את מנת החילוק, ובשורה 13 הורינו להדפיס את שארית החילוק. נתבונן בקלט ופלט של תוכנית זו. נבחר שני ערכים 10 ו-4 אשר נזין כקלט:

```
Type first number:
10
Type second number:
4
Mana = 2
Sherit = 2
```

סדרי עדיפויות בחישובים מתמטיים

כמו באלגברה, כך גם בחישובים הנעשים בפסקל אנו נוהגים לפי סדרי עדיפויות של הפעולות. לדוגמה, פעולות כפל וחילוק קודמות לפעולות חיבור וחיסור. לדוגמה,

```
Y := X + Z / 3;
```

במשוואה זו יחושב הערך Z / 3, ולאחר מכן יחובר אליו הערך שבמשתנה X. אם היינו רוצים לחבר את שני המשתנים X ו-Z ורק אז לחלק את הסכום ב-3, היינו מוסיפים למשוואה **סוגריים**.

```
Y:=(X + Z) / 3;
```

באלגברה למדנו שקיימים שלושה סוגים של סוגריים. הסוג הראשון הוא סוגריים עגולים ״()״, סוגריים מרובעים ״[ ]״, והסוג השלישי סוגריים מסולסלים ״{ }״.

כאשר כותבים ביטוי מתמטי במחשב אנו משתמשים **בסוגריים העגולים בלבד**, כמה פעמים שנרצה. לדוגמה נכתוב את המשוואה הזו:

{X + [Z * 2 + (14 * X + 15) / 3]}

בשפת פסקל נכתוב אותה באופן הבא:

```
      3     2          1                1    23
Y := (X + (Z * 2 + (14 * X + 15) / 3));
```



שימו לב לסגור סוגריים שפתחתם. *בדוגמה לעיל, המספרים שמעל המשוואה מציינים את "זוגות" הסוגריים (הם **אינם** מהווים חלק של הפקודה בפסקל).*

דוגמאות נוספות למשוואות:

```
Y := 2 * X + Z / 4;
X := Y / 4 * Z;
A := (14 + B div 5) * 16;
C := (A + B) div 4;
```

כעת נפתור דוגמה מעשית. עלינו לכתוב תוכנית, אשר קולטת את הערכים הבאים: זמן נסיעה של מכונית (בשעות), מהירות ממוצעת (בקמ"ש), מספר הקילומטרים לליטר ומחיר ליטר דלק. יש לחשב את עלות הנסיעה על פי הנתונים הללו.

נבחן את המשתנים שדרושים לנו:

T - זמן נסיעת המכונית.

V - מהירות המכונית.

S - הדרך שעברה המכונית.

C - מספר הקילומטרים לליטר.

P - מחיר ליטר דלק.

Total - מחיר כולל (עלות הנסיעה).

כעת נוכל לגשת לכתיבת התוכנית:

```
1)  program  Targil;
2)  var
3)      T, V, S, C, P, Total:  Real;
4)
5)  begin
6)      Writeln ('זמן נסיעה:');
7)      Readln (T);
8)      Writeln ('מהירות ממוצעת:');
9)      Readln (V);
10)     Writeln ('מספר קילומטרים לליטר דלק:');
11)     Readln (C);
12)     Writeln ('מחיר ליטר דלק:');
13)     Readln (P);
14)
15)     S := T * V;
16)     Total := (S / C) * P;
17)     Writeln ('ש"ח', Total, 'עלות הנסיעה:');
18) end.
```

נבחן את מהלך ביצוע התוכנית:

בשורה 3 הגדרנו את כל המשתנים הדרושים. בשורות 6 עד 13 אנו קולטים את ארבעת הנתונים הבסיסיים: זמן נסיעה, מהירות ממוצעת, מספר קילומטרים לליטר ומחיר לליטר.

עלינו לחשב תחילה כמה קילומטרים נסעה המכונית. המשוואה שבשורה 15 מבצעת מטלה זו. על ידי כפל מהירות המכונית בזמן הנסיעה מקבלים כמה קילומטרים המכונית נסעה. כדי למצוא את עלות הנסיעה עלינו לחשב את מספר הליטרים שהמכונית צרכה. עושים זאת על ידי חלוקת מספר הקילומטרים שהמכונית עברה במספר הקילומטרים שהיא "נוסעת" לליטר. לדוגמה, אם מכונית צורכת עשרה קילומטרים לליטר, הרי שלנסיעה בת 30 קילומטרים יידרשו 3 ליטרים. לכן, הביטוי S / C מחשב את מספר הליטרים שנדרשו למכונית. אם נכפול את מספר הליטרים שצרכה המכונית במחיר של ליטר אחד, נמצא את עלות הנסיעה הכוללת. הביטוי שבשורה 16 מבצע חישוב זה. תפקיד שורה 17 הוא רק להדפיס את העלות בצירוף הודעה מתאימה.

פרוצדורות ופונקציות

הפקודות בשפת פסקל מתחלקות למספר קבוצות, וביניהן: פרוצדורות ופונקציות. המילה **פרוצדורה** נובעת מהמילה הלועזית **procedure** שמשמעותה **תהליך**; כשמה כן היא. פקודה שהיא פרוצדורה מקבלת ערך או ערכים מסוימים, ועל פיהם היא מבצעת תהליך כלשהו. לדוגמה, הפקודה Writeln היא פרוצדורה, מכיוָן שהיא מבצעת תהליך מסוים על פי הנתונים שנותנים לה. על פי המשפטים, מילים או משתנים הפרוצדורה Writeln יודעת כיצד היא צריכה "להתנהג", כלומר להדפיס פלט על המסך. גם הפקודה Readln נחשבת לפרוצדורה, מכיון שהיא קוראת את הקלט לפי המשתנים שמציינים לה. ניתן לסכם ולומר, כי פרוצדורה היא פקודה המבצעת תהליך על פי הנתונים שאנו מגדירים לה.

מהי **פונקציה** (**function**)? במתמטיקה פונקציה היא פעולה המתבצעת על ערך (מקור) ומחזירה ערך (תמונה) (אסור שתי תמונות למקור אחד).

כתיבת פונקציות בשפת פסקל דומה לכתיבת פונקציות במתמטיקה:

```
x := Abs (y);
```

ככלל, ניתן להגדיר את תבנית השימוש בפונקציה באופן הבא:

```
var := function_name (data);
```

var הוא המשתנה שמקבל את התוצאה.

function_name הוא שם הפונקציה שאנו רוצים להפעיל.

data הוא הנתון (או הנתונים) שאנו נותנים לפונקציה לצורך חישוב או עיבוד. נתונים אלה יכולים להיות קבועים או משתנים, או ביטוי משולב.

הפונקציה Abs (ערך מוחלט)

הפונקציה Abs מחזירה את הערך המוחלט של ערך נתון כלשהו. נזכיר כי ערך מוחלט הוא ערך ללא סימן וכותבים אותו במתמטיקה כך: IXI. לדוגמה, הערך המוחלט של 1+ הוא 1, וגם הערך המוחלט של 1- הוא 1. כיצד, אם כן, משתמשים בפונקציה?

תוכנית דוגמה:

```
1)  program  Dugmma;
2)  var
3)     I:             Integer;
4)
5)  begin
6)     Writeln ('Type an integer value: ');
7)     Readln (I);
8)     Writeln ('Absolute value of ', I, ' is: ', Abs (I));
9)  end.
```

בתוכנית זו אנו מגדירים משתנה שלם (I) בשורה 3. בשורה 6 תדפיס התוכנית הודעה בנוסח "הכנס מספר שלם כלשהו". בשורה 7 תקלוט התוכנית את המספר, ובשורה 8 תדפיס את ערכו המוחלט עם הודעה מתאימה. בהנחה כי קלט התוכנית יהיה 9-, נקבל על המסך:

```
Type an integer value:
-9
Absolute value of -9 is 9
```

הפונקציה Sqr (ערך ריבועי)

הפונקציה Sqr מחשבת את הערך הריבועי (בחזקה של שניים) של נתון כלשהו. לדוגמה, התוכנית הבאה תקלוט מספר ממשי ותדפיס את ערכו הריבועי:

```
program  Square;
var
   R:             Real;

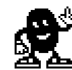
begin
   Writeln ('Type in a real value:');
   Readln (R);
   Writeln ('Square = ', Sqr (R));
end.
```

הפונקציה Sqrt (שורש ריבועי)

התהליך ההפוך של חישוב ערך ריבועי הוא חישוב שורש ריבועי. לשם כך מספקת לנו פסקל את ה**פונקציה Sqrt** שמחשבת שורש ריבועי של ערך נתון. לדוגמה, התוכנית הבאה תחשב את השורש הריבועי של קלט התוכנית:

```
program SquareRoot;
var
   R:          Real;

begin
   Writeln ('Type in a real value:');
   Readln (R);
   Writeln ('Square root = ', Sqrt (R));
end.
```



הערה: *חשוב לציין שאין שורש למספר ממשי שלילי, ולכן הניסיון לחשב שורש ריבועי של מספר כזה יגרום למחשב לעצור ולהודיע על שגיאה.*

הפונקציה Ord (המרת תו לערך אסקי)

לכל תו במחשב יש קוד מספרי המייצג אותו. כדי ליצור אחידות, נקבע תקן המגדיר מספרים המייצגים את התווים. לדוגמה, נקבע כי הערך 65 מייצג את התו A, הערך 66 מייצג את התו B, הערך 38 מייצג התו & וכך הלאה. לכל מספר המייצג תו קוראים **קוד אסקי** (**ASCII Code**) על שם המכון האמריקאי שקבע תקן זה.

כדי להקל על השימוש בקודים אלה, מספקת פסקל פונקציה שתפקידה לקבל תו כלשהו ולהמיר אותו לקוד אסקי. זוהי ה**פונקציה Ord** שאנו עוסקים בה כעת. לשם הדגמה, נבחן את התוכנית הבאה:

```
program Targil;
var
   Ch:         Char;
   X:       Integer;

begin
   Ch := '4';
   X := Ord (Ch);
end.
```

בתוכנית זו הגדרנו שני משתנים. הראשון הוא Ch מסוג תו, והשני הוא X מסוג שלם. בתחילת התוכנית הצבנו במשתנה Ch את התו "4", ובשורה לאחר מכן קיבלנו את ערך אסקי של המשתנה השלם X. כעת אנו יודעים כיצד פונקציה זו פועלת: היא מקבלת תו ומחזירה ערך שלם המייצג את קוד אסקי שלו.

הפונקציה Chr (המרת קוד אסקי לתו)

הפונקציה Chr הפוכה לפונקציה Ord. תפקידה לקבל קוד אסקי ולהמיר אותו לתו המתאים. נדגים זאת בעזרת התוכנית הבאה:

```
program Targil;
const
   I:         Integer  =  68;

var
   Ch:        Char;

begin
   Ch := Chr (I);
end.
```

בתוכנית זו הגדרנו משתנה מאותחל מראש (I) מסוג שלם, שערכו 68. בנוסף הגדרנו גם משתנה מסוג תו בשם Ch. מהלך התוכנית כולל שורה אחת שתפקידה להמיר את הערך שבמשתנה I לתו. תהליך זה נעשה בעזרת הפונקציה Chr, שמקבלת את ערכו של המשתנה I וממירה אותו לתו המאוחסן במשתנה Ch.

הפונקציות Int (בידוד הערך השלם) ו-Trunc (המרת ערך ממשי לשלם)

שפת פסקל כוללת מספר סוגים של פונקציות, כפי שכבר ראינו, ביניהן פונקציות חישוב, כמו למשל, הפונקציה **Sqrt**, אשר מחשבת שורש של ערך נתון ופונקציות המרה של תווים לקוד אסקי ולהיפך. פונקציות ההמרה אינן פונקציות חישוב במובן הפשוט, וכך גם הפונקציה Int. היא עוסקת בערכים ממשיים, אך אינה פונקציה שמחשבת ערכים. כעת נסביר זאת.

תפקיד **הפונקציה Int** להציג את הערך השלם בלבד של מספר ממשי. לדוגמה, אם ניתן לפונקציה את הערך 13.96 היא תחזיר את הערך 13.00. אם ניתן לפונקציה את הערך 9.4623- היא תחזיר את הערך 9.00- וכך הלאה. פונקציה זו "מבודדת" את הערך השלם מתוך הערך הממשי.

הפונקציה Trunc גם כן מבודדת את הערך השלם מתוך ערך ממשי נתון.

ההבדל בין שתי הפונקציות הוא בכך, שהפונקציה Int מקבלת **ערך מסוג ממשי** ומחזירה **ערך מסוג ממשי**. לעומתה, הפונקציה Trunc מקבלת **ערך מסוג ממשי** ומחזירה **ערך מסוג שלם**.



זכרו: *בשני המקרים מקבלים את הערך* ***השלם*** *של הערך הנתון בלבד.*

הפונקציה Round (עיגול ערך ממשי)

למדנו שהפונקציה Trunc מאפשרת להמיר ערך ממשי לערך שלם. לעיתים אנו רוצים לעגל ערך, ולא לקצץ את החלק העשרוני של המספר. כדי לעשות זאת צריך להתחשב בשבר לצורך ההמרה.

לדוגמה, נעגל את הערך 4.49 לערך 4, מכיון שהשבר קטן מ-0.5. בדומה, את הערך 4.58 נעגל לערך 5 מכיוָן שהשבר שווה (או גדול במקרים אחרים) ל-0.5.

כדי לעשות זאת נשתמש ב**פונקציה Round** לפי הדוגמה בתוכנית הבאה:

```
program RoundReal;
var
   R:          Real;

begin
   Writeln ('Type in a real value:');
   Readln (R);
   Writeln ('Rounded value: ', Round (R));
end.
```

בתוכנית זו הגדרנו משתנה ממשי R. במהלך התוכנית תודפס הודעה המבקשת מהמשתמש להכניס ערך ממשי כלשהו (הערך נקלט במשתנה R). אחר כך התוכנית תדפיס את הערך לאחר שעבר תהליך עיגול ותציג הודעה מתאימה.

הפונקציה Frac (בידוד שבר עשרוני)

דנו זה עתה בתפקיד הפונקציה Int. מטרתה לבודד את הערך השלם מתוך ערך ממשי נתון. קיימת פונקציה נוספת ה"משלימה" אותה. הכוונה היא, שקיימת פונקציה שתפקידה לבודד את השבר, בדומה לאופן שהפונקציה Int מבודדת את החלק השלם. לפונקציה זו קוראים **Frac**. הפונקציה מקבלת ערך ממשי כלשהו ומחזירה ערך ממשי המורכב אך ורק **משבר עשרוני**. לדוגמה, אם ניתן לפונקציה את הערך 11.059 נקבל את הערך 0.059.

נדגים זאת בעזרת התוכנית הבאה:

```
program  RealValues;
var
   R:          Real;

begin
   Writeln ('Type in a real value:');
   Readln (R);
   Writeln ('Main value: ', Int (R));
   Writeln ('Fraction: ', Frac (R));
end.
```

בתוכנית זו הגדרנו משתנה ממשי R. בתחילת התוכנית מציגים הודעה המבקשת מהמשתמש להכניס ערך ממשי שלאחר מכן נקלט לתוך המשתנה R. בסופה של התוכנית אנו מדפיסים את ערכו השלם של הערך (בעזרת הפונקציה Int) ואת ערך השבר (בעזרת הפונקציה Frac).

הפונקציה Random (יצירת מספר אקראי)

אם נפעיל את הפונקציה **Sqr** על הערך 3 מספר פעמים, נקבל תמיד את הערך 9, מכיון שתמיד נפעל על אותו ערך בסיסי.

הפונקציה **Random** מחזירה ערכים **שונים** בכל הפעלה שלה: אלה הם ערכים אקראיים מתוך קבוצת ערכים המוגדרת בארגומנט הנמסר לפונקציה.

הפעלת הפונקציה

```
random (n)
```

n - מספר שלם.

הפונקציה **מחזירה** מספר שלם בתחום: N < ערך מוחזר >= 0

דוגמה: הדפסת מספרים אקראיים בתחום 1 עד 49 (כמו בלוטו).

```
begin
   for i: = 1 to 6 do
      writeln(random(49)+1);
end.
```

בקטע זה, התוכנית "מגרילה" 6 מספרים שמודפסים על ידי הפרוצדורה Writeln.



שימו לב: *המספר שמוגרל הוא בתחום 0 עד 48 (כולל שני הערכים הקיצוניים) ולכן, לאחר הגרלת המספר אנו מחברים 1 למספר האקראי שהתקבל, כדי לקבל מספר בתחום 1-49 (בהתאם לדרישתנו).*

שאלה: האם כל 6 המספרים שיוגרלו יהיו שונים?

תשובה: לא! בכל פעם שהפונקציה מופעלת יש סיכוי שווה לכל אחד מהמספרים בתחום 0 עד 48, וייתכן מקרה קיצוני שאינו סביר מבחינה סטטיסטית, שבו כל 6 המספרים האקראיים יהיו זהים.

הפרוצדורה Randomize

הפונקציה Random מחזירה תמיד את אותה קבוצת ערכים מתוך קבוצה מוגדרת (אין זו אקראיות אמיתית). על כן, רצוי לרשום קריאה לפרוצדורה Randomize, ללא פרמטרים כלשהם. הפרוצדורה הזו יוצרת שינוי באופן הגרלת המספרים האקראיים כשמפעילים את הפונקציה Random.

נכתוב תוכנית לדוגמה:

```
program exe;
   var
   i,x,y,ans:integer;

   begin
      randomize;
      for i:= 1 to 9 do
      begin
         x:=random(10)+1;
         y:=random(10)+ 1;
         writeln (x,' * ',y,'  = ');
         readln(ans);
         if ans = x * y then
            writeln('good')  else
            writeln('wrong');
      end;
   end.
```

תוכנית זו "מייצרת" 9 תרגילי כפל (המספרים שיוגרלו הם בתחום 1 עד 10) התרגיל הכולל את המספרים שהוגרלו מוצג למפעיל התוכנית, מתקבל קלט תשובה לתרגיל ונבדקת נכונות התשובה.

אתגר: הוסיפו לתוכנית הודעות למשתמש, וכן הוסיפו אפשרות למתן ציון לתלמיד לאחר סדרת התרגילים.

תרגילים

תרגיל 2.1

כתבו תוכנית שקולטת שלושה ערכים שלמים (A, B, C) ומחשבת את הממוצע.

תרגיל 2.2

כתבו תוכנית המקבלת כקלט שלושה ערכים שלמים לתוך המשתנים A, B ו-C. התוכנית תחשב את השארית מחלוקת A בסכום B+C.

תרגיל 2.3

כתבו תוכנית הקולטת משקל בטונות ומדפיסה אותו בגרמים.

תרגיל 2.4

כתבו תוכנית הקולטת שלושה ערכים ממשיים ומדפיסה את סכומם, מכפלתם והממוצע שלהם.

תרגיל 2.5

כתבו תוכנית הקולטת ערך שלם ומדפיסה את ערכו בסימן ההפוך. לדוגמה, הכנסת המספר 4 תגרום למחשב להדפיס את המספר 4-, הכנסת המספר 9- תגרום למחשב להדפיס את המספר 9 וכך הלאה.

תרגיל 2.6

כתבו תוכנית הקולטת מספר בעל ספרה אחת או שתי ספרות ומדפיסה את ערכי היחידות והעשרות של המספר.

(**רמז:** יש להשתמש באופרטורים mod ו-div).

תרגיל 2.7

כתבו תוכנית הקולטת ערך, מחשבת ומדפיסה את הערך הריבועי שלו ואת השורש הריבועי. כדי לא לגרום לבעיות, על התוכנית לחשב תחילה את ערכו המוחלט של המספר הנקלט. (**רמז**: את משתנה הקלט יש להגדיר כמשתנה ממשי).

תרגיל 2.8

כתבו תוכנית הקולטת תו בודד ומדפיסה את ערך אסקי שלו.

תרגיל 2.9

כתבו תוכנית הקולטת תו בודד ומדפיסה את התו הבא בטבלת אסקי.

תרגיל 2.10

כתבו תוכנית הקולטת ערך ממשי ומדפיסה את החלק העשרוני שלו.

(**רמז**: לשם כך יש להשתמש בפונקציה Frac).

תרגיל 2.11

כתבו תוכנית הקולטת את שלושת המימדים של קוביה (גובה, אורך ורוחב) ומחשבת את הנפח שלה.

תרגיל 2.12

כתבו תוכנית הקולטת ערך ממשי ומיישמת את הפעולה של הפונקציה Frac, בתהליך שיתן את אותה התוצאה ללא שימוש בפונקציה זו.

תרגיל 2.13

כתבו תוכנית הקולטת ערך שלם שאינו שווה ל-0 ומדפיסה את סימנו בלבד. לדוגמה, אם ייקלט ערך שלילי התוכנית תדפיס את הערך 1-, ואם ייקלט מספר חיובי התוכנית תדפיס 1. (**רמז**: יש להשתמש בפונקציה Abs).

תרגיל 2.14

כתה בת 37 תלמידים יצאה לטיול. במהלך הטיול התבקשו התלמידים להתחלק לקבוצות של חמישה תלמידים לכל קבוצה. כתבו תוכנית המדפיסה את מספר הקבוצות שבהן יש חמישה תלמידים ואת מספר התלמידים שנשארו ללא קבוצה.

(**רמז**: יש להשתמש באופרטורים mod ו-div).

פרק 3

אלגברה בוליאנית

מהי אלגברה בוליאנית?

כאשר עסקנו בפרק הראשון בסוגים שונים של משתנים, פגשנו בסוג מיוחד של משתנה בשם **משתנה בוליאני** (**Boolean variable**). משתנה מסוג זה מייצג אך ורק שני ערכים: **כן** ו**לא** (Yes/No). אם היה באפשרותנו לרדת לרמה הבסיסית של המחשב ולנתח את מבנהו של משתנה זה, היינו רואים כי זהו למעשה משתנה שלם, המכיל את הערך "0" או "1". הערך "0" מייצג **לא** והערך "1" מייצג **כן**. נזכיר גם, כי **כן** מייצג את המילה **True** ו**לא** מייצג את המילה **False**. **האלגברה הבוליאנית** מטפלת בפסוקים לוגיים וחישוב פסוקים לוגיים. **פסוק לוגי** הוא ביטוי פשוט או מורכב המקבל ערך "אמת" או "שקר". פעולות על פסוקים לוגיים מתבצעות באמצעות **אופרטורים לוגיים**. נגדיר את המאפיינים הבסיסיים של אלגברה זו:

א) המילה **False** (**שקר**) מגדירה את המצב 0, והמילה **True** (**אמת**) מגדירה את המצב 1.

ב) 0 הוא ההיפך של 1 (1 הוא ההיפך של 0).

ג) קיימות שלוש פעולות בסיסיות: חיבור, כפל והיפוך.

ד) פעולת חיבור לוגי:

0 + 0 = 0
0 + 1 = 1
1 + 0 = 1
1 + 1 = 1

ה) פעולות כפל לוגי:

0 * 0 = 0
0 * 1 = 0
1 * 0 = 0
1 * 1 = 1

ו) פעולת היפוך לוגי:

פעולת ההיפוך הופכת את מצב הערך, כלומר מצב True הופך למצב False ולהיפך.

כיצד כל זה מתקשר למחשב?

לאלגברה הבוליאנית יש שימוש נרחב בתחומי המחשוב. למעשה, זוהי צורת העבודה של המחשב. המעבד הראשי המפעיל את המחשב מבין אך ורק את שני המצבים "0" ו-"1". מכיוָן שזהו המצב, לא היה ניתן להזניח נושא זה בשפת פסקל, שבה קיימים האופרטורים הלוגיים שבאמצעותם ניתן לבנות תנאים מורכבים.

האופרטורים OR (חיבור לוגי), AND (כפל לוגי) ו-NOT (היפוך לוגי)

נגדיר שלושה משתנים A, B ו-C כמשתנים בוליאנים. "נחבר" את A ו-B כך שהתוצאה תתקבל ב-C. נוכל לכתוב זאת כך:

C := A or B;

כלומר, ישנם שני מצבים המיוצגים על ידי A ו- B. **פעולת OR** ("או") גורמת לכך שב-C נקבל מצב "אמת" (1) **אם לפחות אחד** מהמשתנים A או B ייצג מצב 1. ובעברית פשוטה: אם ב-A או ב-B היה "1" - נקבל ב-C את המצב "1".

באותו אופן, אם נרצה לכפול את A ו-B כך שהתוצאה תתקבל ב-C נכתוב:

C := A and B;

אנו מבקשים לקבל ב-C מצב שנמצא **גם** ב-A **וגם** ב-B. כלומר רק אם גם ב-A **וגם** ב-B יש "1" נקבל ב-C את המצב "1". בכל מצב אחר נקבל ב-C את המצב "0".

הפעולה הבסיסית האחרונה היא "**היפוך מצב**", ולשם כך אנו משתמשים באופרטור NOT. לדוגמה:

A := not A;

ביטוי זה יגרום למשתנה A **להפוך** את מצבו. אם היה קיים במשתנה הערך "True", הוא יהיה כעת "False" ולהיפך. אם ב-A ישנו המצב "0", ונכתוב את הביטוי הזה,

```
A := not A;
```

נקבל שערך המשתנה הפך ל-"1".

כמו בפעולות אריתמטיות, גם בפעולות לוגיות יש **סדר עדיפות** לביצוע:

NOT - בעדיפות הגבוהה ביותר, אחריה AND, ולבסוף - OR.

ביטויי שוויון ואי-שוויון

במתמטיקה קיימים שני סוגים של תבניות פסוק: משוואות ואי-שוויונים. משוואות שוויון מגדירים בעזרת התו "=". אי-שוויונים מגדירים על ידי חמישה ביטויים שונים: גדול (>), קטן (<), גדול או שווה (≥), קטן או שווה (≤) אין שוויון (≠). בפסקל מסמנים באופן הבא את ששת הסמלים של שוויון ואי-שוויון:

הביטוי	הסמן
שווה	=
לא שווה	<> (התו < ולאחריו >)
גדול	>
קטן	<
גדול או שווה	>=
קטן או שווה	<=

כעת אנו יכולים לחזור לאלגברה הבוליאנית ולאופן היישום שלה בפסקל. נתבונן בביטוי הבא, שבו המשתנה **A מוגדר כמשתנה בוליאני**:

```
A := 9 > 5;
```

במבט ראשון, ביטוי זה ייראה לנו כטעות. ננסה לבחון את המצב "מנקודת הראות" של המחשב. כאשר המחשב נתקל בביטוי 9 > 5 הוא משווה בין שני הערכים. מכיוָן ש-5 קטן מ-9, המחשב מוצא כי הביטוי הזה **נכון**, ועל כן הוא מציב במקומו את הערך "True". כעת אנו יכולים להבין כי המשתנה A מקבל את הערך "True". באותו אופן, אם היה על המחשב לבחון את הביטוי הבא:

```
A := 2 < 1;
```

היינו מקבלים במשתנה A את הערך "False", מכיוָן שביטוי זה **לא נכון**. כלומר, התשובה היא "לא". התוכנית הבאה מדגימה שימוש בביטוי דומה:

```
program Dogma;
var
   A, B:        Integer;

begin
   Writeln ('Type in two integer values (A, B):');
   Readln (A, B);
   Writeln (A = B);
end.
```

בתוכנית זו הגדרנו שני משתנים שלמים. בשתי השורות הראשונות של התוכנית אנו מדפיסים הודעה המבקשת מהמשתמש להכניס שני ערכים שלמים הנקלטים למשתנים A ו-B. בשורה האחרונה של התוכנית אנו מציגים את התוצאה של הביטוי A = B. אם שני הערכים זהים, התוכנית תדפיס את המילה "TRUE", ואם לא - היא תדפיס "FALSE".

לאחר שהבנו את האופרטורים הללו, נשלב גם את האופרטורים OR ו-AND. נתבונן בדוגמה הבאה (בהנחה שהמשתנים A, B ו-C מוגדרים כשלמים ו-D כמשתנה בוליאני):

```
D := (A < B) and (A = C);
```

את הביטוי הזה ניתן להבין מקריאה בלבד. נסביר אותו: **אם** ערך המשתנה A קטן מערך המשתנה B, **וגם** ערך המשתנה A שווה לערך המשתנה C, **אז** המשתנה D יקבל את הערך "True". בכל מצב אחר יתקבל הערך "False" במשתנה זה. באותו אופן ניתן להשתמש באופרטור OR ("או"):

```
D := (A < B) or (A = C);
```

במשתנה D יתקבל הערך "True" כאשר ערך המשתנה A קטן מערך המשתנה B, **או** כאשר ערך המשתנה A שווה לערך המשתנה C.

תרגילים

תרגיל 3.1

כתבו תוכנית הקולטת שלושה ערכים שלמים. התוכנית תדפיס "True" אם שניים מתוך שלושת הערכים שווים, או "False" אם אין שני ערכים שווים.

תרגיל 3.2

כתבו תוכנית הקולטת שלושה ערכים שלמים. התוכנית תדפיס "True" אם הערכים שהתקבלו הם בסדר עולה (הראשון קטן מהשני, והשני קטן מהשלישי).

תרגיל 3.3

כתבו תוכנית הקולטת שני ערכים שלמים. התוכנית תדפיס "True" אם שני המשתנים שווים ל-0. אם לא, התוכנית תדפיס את הערך "False".

פרק 4

התניות ולולאות

הקדמה

בעזרת הידע שצברנו עד כה עלה בידינו ליצור תוכניות חישוב שונות. אף על פי כן, ניתן להבחין במגבלה בולטת לעין: לא יכולנו להגדיר תנאים לפעולת התוכנית. לדוגמה, אנו רוצים להציג בפני המשתמש את ההודעה "האם ברצונך להמשיך (כן/לא)?", לקרוא את התשובה ולפעול בהתאם.

אנו יודעים כיצד להדפיס את ההודעה, לקרוא את הקלט מהמשתמש, אך אין אנו יודעים כיצד לפעול על פי התשובה, כלומר - להמשיך או להפסיק. נבחן דוגמה נוספת: אנו רוצים לכתוב תוכנית המבקשת מהמשתמש להכניס מספר מ-0 עד 10. אם המשתמש מכניס מספר שגוי, התוכנית תדפיס הודעת שגיאה ותבקש מספר נוסף. גם כאן אנו יודעים כיצד להדפיס הודעה ולקרוא נתונים, אך אין אנו יכולים לבצע עמם דבר.

בפרק זה נעסוק בהתניות ולולאות. נלמד את הדרך שבה מציגים למחשב "תנאי" לבדיקה, ועל-פי תוצאת הבדיקה נקבע המשך **מהלך ביצוע התוכנית**. בנוסף, נלמד גם על הלולאות - הדרכים **לחזור** על סדרת פקודות על פי תנאים שונים.

התניות

התניה היא הגדרה למצב שבו אנו מציגים למחשב **תנאי** (**Condition**). אם התנאי מתקיים, יתבצע תהליך מסוים שקבענו מראש. אם התנאי אינו מתקיים, לא יתבצע התהליך הזה, או לחילופין יתבצע תהליך שונה, ולאחר מכן התוכנית תמשיך במהלכה. את תבנית ההתניה ניתן לכתוב כך:

if [expression] **then** [statement];

בעברית ניתן לתאר זאת כך: **אם** (**if**) [ביטוי התנאי] **אז** (**then**) [פעולה]. כלומר, **אם** התנאי מתקיים **אז** תתבצע הפעולה.

לדוגמה:

```
Readln (A);
if A = 0 then Writeln ('ZERO');
```

בקטע תוכנית זו אנו קולטים ערך מספרי לתוך המשתנה A. בשורה שלאחר מכן אנו מציבים למחשב תנאי: אם הערך במשתנה A שווה ל-0, אז הדפס את המילה ZERO, בכל מקרה אחר כאשר לא יתקבל הערך 0, לא תודפס ההודעה.

כעת נשלב את כללי האלגברה הבוליאנית שלמדנו:

```
Readln (A);
if (A > 0) and (A < 10) then Writeln ('OK');
```

כאן אנו קולטים ערך מספרי לתוך המשתנה A. בשורה השנייה אנו מציבים תנאי: אם ערך המשתנה A גדול מ-0 וקטן מ-10, אז תודפס ההודעה "OK". במילים אחרות, ניתן לומר כי כאשר מתקבל ביטוי שהוא נכון ("True") אזי מתבצעת הפעולה שכתובה לאחר then.

כדי לנצל את הפוטנציאל שקיים בביטוי התניה, יש הרחבה של מבנה זה. בנוסף לתבנית הבסיסית שראינו יש תבנית מורחבת:

```
if [expression] then
  [statement1]
else
  [statement2];
```

בעברית ניתן להגדיר זאת כך:

אם (**if**) **ביטוי התנאי** [**expressions**] מתקיים **אז** (**then**) יש לבצע את קטע התוכנית statement1, **אחרת** (**else**) יש לבצע את statement2.

את ביטוי התנאי כתבנו כפי שלמדנו, אך כאן אנו מבצעים את קטע התוכנית statement1 **אם** התנאי **מתקיים**, ואם התנאי **לא מתקיים** - כלומר, אחרת (else) - מבצעים את קטע התוכנית statement2. לאחר ביצוע אחד משני קטעי התוכנית על פי בדיקת התנאי, התוכנית תמשיך בפקודה שלאחר הפקודה if.

באנגלית מכנים ביטוי זה בשם: **if-then-else**, ובעברית: **אם-אז-אחרת**.

כעת אנו יכולים להרחיב את הדוגמה הראשונה שעסקנו בה:

```
Readln (A);

if A = 0 then
   Writeln ('ZERO')
else
   Writeln ('NOT ZERO');
```

בשורה הראשונה התוכנית קולטת מספר המאוחסן במשתנה A. בשורה שלאחר מכן אנו מציבים התניה: אם הערך שבמשתנה A שווה ל-0 תודפס ההודעה ZERO, אחרת (אם לא שווה ל-0) תודפס ההודעה NOT ZERO. התוספת else חוסכת מאתנו את הצורך בהתניה נוספת עבור המצב השני (המצב בו ערך המשתנה A אינו שווה ל-0) ויוצרת תוכנית מובנת יותר.



הערה: *יש לשים לב כי בסוף הפקודה הראשונה שמופיעה אחרי if (במקרה שלנו ("ZERO") Writeln) אין שמים את התו נקודה-פסיק (;), מכיון שמשפט התנאי* ***לא הסתיים****. התו נקודה פסיק (;) יופיע* ***בסיום*** *המשפט if בלבד.*

נבחן דוגמה נוספת:

```
1)  program Dogma;
2)  var
3)     Ch:          Char;
4)
5)  begin
6)     Writeln ('Type in a single char:');
7)     Readln (Ch);
8)     if  Ch = 'Y'  then
9)        Writeln ('YES');
10)    if  Ch = 'N'  then
11)       Writeln ('NO');
12) end.
```

בתוכנית זו הגדרנו משתנה Ch מסוג תו. בשורה 6 אנו מציגים הודעה המבקשת מהמשתמש להקיש תו בודד. בשורה 7 התוכנית קוראת את התו הזה ומאחסנת אותו במשתנה Ch. ההתניה שבשורה 8 קובעת: אם תוכן המשתנה Ch (התו שנתקבל על ידי המשתמש) הוא "Y", תודפס המילה YES. בשורה 10 קיימת התניה דומה: אם תוכן המשתנה Ch הוא "N" - תודפס המילה NO.

לפני שנבחן דוגמה נוספת, נזכיר כי לכל תו יש קוד אסקי (דנו בנושא זה כאשר עסקנו בפונקציות Chr ו-Ord בפרק 2). בשל כך, על פי טבלת קוד אסקי האות "A" נמצאת לפני האות "B", והאות "Z" נמצאת אחרי האות "Y". בדוגמה הבאה ננצל תכונה זו:

```
1)  program Dogma;
2)  var
3)    Ch:        Char;
4)
5)  begin
6)    Writeln ('Type in a single char: ');
7)    Readln (Ch);
8)    if (Ch >= 'A') and (Ch <= 'Z') then
9)       Writeln ('Value between A to Z !!!');
10) end.
```

בתוכנית זו הגדרנו משתנה Ch מסוג תו. בשורה 7 התוכנית קולטת תו בודד לתוך המשתנה Ch. בשורה 8 קיימת התניה שננתח את מרכיביה. בחלק הראשון של ההתניה נשאלת שאלה: האם התו המאוחסן במשתנה Ch גדול או שווה ל-A. לכל תו יש קוד מספרי המייצג אותו, ולכן ניתן לערוך השוואות בין תווים. במקרה שלנו, אנו שואלים אם קוד אסקי של התו במשתנה Ch גדול או שווה לקוד אסקי של התו A. במקרה השני אנו שואלים שאלה דומה: האם קוד אסקי של התו במשתנה Ch קטן מקוד אסקי של המשתנה Z. את שתי ההתניות הללו אנו כותבים יחד, וכך מקבלים התניה מורכבת (שורות 8-9): **אם** התו שב-Ch גדול או שווה ל-"A" **וגם** קטן או שווה ל-"Z" (כלומר בתחום של A עד Z) **אז** התוכנית תדפיס את ההודעה "Value between A to Z !!!".

כעת נוכל להבין את החשיבות של קוד אסקי, שבעזרתו ניתן לקבוע היכן נמצא כל תו בטבלה ומהו "סדר" התווים. לדוגמה, התו A קטן מבחינה ערכית מהתו O וגם התו 0 קטן מהתו 8.



הערה: *יש לציין כי ערך אסקי של התו "a" אינו שווה לערך אסקי של התו "A". לתו "a" יש ערך גבוה מזה של "A". כלומר, יש אבחנה ברורה בין ערכי התווים של אותיות גדולות וקטנות באנגלית.*

לולאות

מבנה תכנות בסיסי נוסף הוא **הלולאה** (**loop**). הלולאה כשמה כן היא - **תהליך החוזר על עצמו**. התהליך הוא קבוצת פקודות. קיימים שני סוגים של לולאות:

- לולאה קבועה.
- לולאה מותנית.

לולאה קבועה היא תהליך החוזר על עצמו מספר קבוע של פעמים, כפי שהמתכנת קובע מראש. בשפת פסקל, כמו בשפות אחרות, אנו משתמשים במבנה הבא:

for var_name := min **to** max **do** [statement];

ניתן לנסח זאת בעברית בצורה הבאה:

בזמן (**for**) שהמשתנה השלם var_name נע בתחום שבין min **עד** (**to**) max, **בצע** (**do**) את ה**הוראה** ([**statement**]).

לדוגמה, אם ברצוננו להדפיס את כל הערכים מ-1 עד 10, נכתוב:

```
for I := 1 to 10 do Writeln (I);
```

בתחילה, המשתנה I (משתנה שלם) יקבל את הערך 1. לאחר הדפסת הערך (בעזרת הפקודה Writeln) יוגדל ערכו של המשתנה I ב-1, ושוב יודפס ערכו. תהליך זה יימשך עד אשר I יגיע לערכו העליון (10).

נציג דוגמה נוספת: אנו רוצים לכתוב תוכנית שתשאל את המשתמש איזה תחום ערכים הוא רוצה להציג על המסך. עם הידע שהיה לנו עד כה לא היה באפשרותנו ליצור תהליך שכזה, אך כעת אנו יכולים לעשות זאת:

```
1) program Dugma;
2) var
3)   L, H, Counter:  Integer;
4)
5) begin
6)   Writeln ('Type the low and the high limits (L, H):');
7)   Readln (L, H);
8)   for  Counter := L  to  H  do  Writeln (Counter);
9) end.
```

בתוכנית זו הגדרנו שלושה משתנים שלמים: L שיגדיר את התחום התחתון, H שיגדיר את התחום העליון, ו-Counter - שישמש כמונה. בשורה 6 התוכנית מדפיסה בקשה להכנסת שני מספרים שיגדירו את התחום התחתון והעליון.

בשורה 7 התוכנית קולטת את שני הערכים הללו לתוך המשתנים L ו-H. בשורה 8 הגדרנו לולאה שתגרום למשתנה Counter "לנוע" בתחום בין L ל-H, כך שכל פעם יודפס ערכו של המונה Counter.

נבחן עתה מצב נוסף של הלולאה:

```
for I := 3 to 1 do ... ;
```

כאן מובאת בפנינו דוגמה שבה אנו מבקשים מהמחשב למנות מ-3 עד 1. מכיוָן שהתחום התחתון (3) **גדול** בערכו של התחום העליון (1), הלולאה לא תתבצע אפילו פעם אחת. בכל זאת, מה יקרה כאשר נהיה חייבים למנות בסדר יורד, לדוגמה מ-9 ל-4? לשם כך נגדיר גירסה נוספת ללולאה for. **במקום** המילה **to** נכתוב **downto**, שפירושה "בירידה עד...". כעת, אם נרצה למנות בסדר יורד מ-0 עד 4-, לדוגמה, נוכל לכתוב:

```
for I := 0 downto -4 do  ... ;
```

לולאות מותנות

תפקיד הלולאה להקל עלינו בתכנות קטעי תוכנית שחוזרים על עצמם. במקרים רבים צריכים להשתמש בלולאות שאינן קבועות. לדוגמה, אנו סורקים את הזיכרון כדי למצוא נתון מסוים, וכאשר נמצא את מבוקשנו נוכל לסיים את הלולאה. דנו לא מכבר בעובדה שהלולאה **for** מתחילה למנות מהערך התחתון עד הערך העליון ללא הפסקה. בהנחה שנשתמש בלולאה זו לסריקת הזיכרון גם אם היינו מוצאים את הנתון המבוקש בתחילת הסריקה, היה עלינו להמתין שהלולאה תסיים את עבודתה. במקרים רבים דבר זה גוזל מזמנו של המחשב (וגם מזמננו, בעבודות גדולות) והוא מיותר לחלוטין, ולפעמים אף מפריע. ניתן אמנם להפסיק לולאת for באמצע התהליך, אך לא רצוי. לשם כך נשתמש בלולאות מסוג אחר. לולאות אלו **פועלות על פי תנאי.**

לדוגמה, אנו רוצים לקרוא לולאה שתקרא ערכים שלמים עד אשר המשתמש יכניס ערך שלילי. לשם כך נכתוב לולאה הפועלת על פי תנאי:

```
repeat
   Readln (I);
until I < 0;
```

בשלוש השורות הללו הגדרנו לולאה החוזרת על עצמה עד אשר מתקבל ערך שלילי (until I < 0).

כיצד נעשה הדבר הזה? המילה **repeat (חזור)** מציינת למחשב כי עתה התחלנו לולאה. בעזרת הפקודה Readln אנו קולטים ערכים שלמים לתוך המשתנה I. המילה **until** (**עד אשר**) מציינת כי הלולאה תחזור על עצמה **עד אשר** המשתנה I יהיה קטן מ-0, כלומר - יכיל ערך שלילי. כך אנו יכולים להגדיר לולאה שתפעל **כל עוד התנאי לא מתקיים**. מקובל לכנות לולאה כזו בשם **לולאת repeat-until.**

נבחן את הדוגמה הבאה:

```
1)  program Dugma;
2)  var
3)     Total:     Integer;

4)  begin
5)     Total := 0;
6)     repeat
7)     Writeln('Enter a number(0 will end the loop):');
8)     Readln (I);
9)     Total := Total + I;
10)    until  I  =  0;
11)    Writeln ('Total  =  ',  Total);
12) end.
```

בתוכנית זו הגדרנו שני משתנים: Total שישמש כמונה ויכיל את הסכום הכולל, ו-I שישמש לקליטת ערכים מהמשתמש. בשורה 5 אנו מאפסים את המשתנה Total. בשורה 6 אנו פותחים את לולאת ההתניה repeat שמסתיימת בשורה 10. בשורה 7 אנו מבקשים מהמשתמש להכניס ערך כלשהו, או את המספר 0 שיציין את סיום הלולאה.

בשורה 8 אנו קולטים את מהמשתמש ערך ומכניסים אותו למשתנה I ומוסיפים זאת לערך הכולל שמנוהל על ידי המשתנה Total. עתה, בשורה 10 הצבנו תנאי: על הלולאה לחזור על עצמה עד אשר הערך שהתקבל מהמשתמש שווה ל-0. בצורה זו אנו יכולים להכניס מספר כלשהו של ערכים, עד אשר נכניס 0 שיגרום ללולאה להסתיים.

בנוסף ללולאת ההתניה ... repeat ... until קיימת לולאה נוספת הפועלת באופן דומה. זוהי **לולאת while** (**"כל עוד..."**). לולאה זו מקיימת את התנאי ההפוך מזה של הלולאה repeat-until. היא פועלת **כל עוד** מתקיים תנאי מסוים, בזמן שהלולאה repeat-until מתקיימת כל עוד התנאי **לא** מתקיים.

את מבנה הלולאה while ניתן להגדיר כך :

while [condition] **do** [statement];

[condition] התנאי שצריך להתבצע.

[statement] התהליך שימשיך להתבצע, כל עוד מתקיים התנאי שהוצב ללולאה.

דוגמה :

```
I := 1;
while I > 0 do Readln (I);
```

בקטע תוכנית זה יצרנו לולאה הקוראת ערכים שלמים מהמשתמש. הלולאה תמשיך לפעול **כל עוד** הנתון המתקבל גדול מ-0. לשם כך היה עלינו לתחל את המשתנה I, כדי שנוכל להיות בטוחים שהתנאי I > 0 יתקיים לפחות פעם אחת.

תנאי לולאות מורחבים

עד עתה עסקנו בהתניות ולולאות במבנה המקוצר שלהן. לבד מהלולאה repeat-until ניתן היה לבצע בכל ההתניות והלולאות פקודה אחת לכל מחזור. עם זאת, ברוב המקרים יהיה עלינו לבצע כמה פעולות בכל מחזור של הלולאה. לדוגמה, אם נרצה לסכם חמישה ערכים, נוכל להשתמש בלולאה for. אך אנו נתקלים בבעיה, המבנה שאנו מכירים נראה כך :

```
for Counter := 1 to 5 do Readln (I);
```

הלולאה אכן תוכל לקרוא חמישה ערכים, אך לא יהיה באפשרותה לעשות אתם דבר. לשם כך ניתן ליצור מבנה הנקרא **בלוק פקודות**. זהו רצף של פקודות המוגדר בין שני תחומים על ידי המילים **begin** ו-**end**, אשר תוחמים למעשה קטע תוכנית. לדוגמה, את בעיית לולאת for נפתור כך :

```
Total := 0;
for  Counter := 1 to 5 do
   begin
      Readln (I);
      Total := Total + I;
   end;
```

בדרך זו הלולאה תבצע את בלוק הפקודות חמש פעמים. הבלוק כולל את הפקודה Readln לקריאת נתון מהמשתמש, והוספתו למשתנה Total.

בשיטה זו ניתן לפתור גם בעיות מורכבות יותר. לדוגמה, עתה יהיה ניתן לכתוב תוכנית המציגה רשימה של אפשרויות על המסך. התוכנית תקלוט את בחירת המשתמש ועל ידי התניות, היא תוכל לבצע תהליכים רחבים יותר.

דוגמה נוספת:

נכתוב תוכנית הקולטת עשרה ערכים שלמים ומדפיסה את הערך הגדול ביותר שנקלט. כל הערכים הנקלטים הם חיוביים.

```
1)  program  Dogma;
2)  var
3)     I, H, Counter: Integer;
4)
5)  begin
6)     H := 0;
7)     for  Counter := 1  to 10 do
8)         begin
9)             Writeln ('Type in an integer value:');
10)            Readln (I);
11)            if  I > H  then H := I;
12)        end;
13)
14)    Writeln ('Highest value: ', H);
15) end.
```

בתוכנית זו הגדרנו שלושה משתנים שלמים: I שישמש לקבלת הקלט מהמשתמש, H שמייצג את הערך הגבוה ביותר שנקלט, ו-Counter המשמש כמונה לולאה.

בשורה 7 אנו פותחים את הלולאה for שתחזור על עצמה עשר פעמים. בשורה 9 מציגים למשתמש הודעה בה הוא מתבקש להכניס ערך שלם, ובשורה 10 הערך נקלט לתוך המשתנה I.

בשורה 11 הצבנו תנאי: **אם** המשתנה I (שזה עתה נקלט לתוכו ערך חדש) מכיל ערך הגבוה מזה שקיים במשתנה H (המייצג את הערך הגדול ביותר), **אז** אנו מעדכנים את H מפני שנמצא ערך חדש גבוה יותר מזה שהיה בו קודם לכן. בסופה של הלולאה (שורה 14) תודפס הודעה בצירוף הערך הגבוה ביותר שהוזן על ידי משתמש.

שימו לב: *אם כל המספרים שנקלטו הם שליליים, נקבל שהמספר הגדול הוא 0. הציעו דרך לתיקון התוכנית.*

לולאות מקוננות

נושא מיוחד בתחום הלולאות הוא **קינון** (**Nesting**). אין זה סוג חדש של לולאה, אלא טכניקה מיוחדת שמנצלת את תכונות הלולאה. השם "קינון" נובע מן העובדה כי לולאה אחת ממוקמת בתוך לולאה אחרת, כך שהן (קשורות) זו בזו.

לפנינו תרגיל:

עלינו לכתוב תוכנית הקולטת את ציוניהם של תלמידי כיתה ט', ומחשבת את הממוצע. לאחר מכן התוכנית קולטת את ציוניהם של בני כיתה י', ומחשבת את הממוצע, וכך הלאה עד כיתה י"ב.

במקרה פשוט נוכל להשתמש בלולאה עבור כל כיתה. אך, מה יקרה אם יציבו בפנינו את הקריטריון הבא: בנוסף לכך, על התוכנית לשאול את המשתמש לכמה כיתות יש נתונים. בעיה זו מגבילה אותנו, במקרה וחשבנו להגדיר לולאה עבור כל כיתה. לכן, בטכניקה של לולאות מקוננות אנו משיגים שתי מטרות: תחילה, אנו פותרים את בעיית המספר המשתנה של הכיתות וגם חוסכים כתיבה חוזרת של לולאות.

נבחן את התוכנית הבאה:

```
program Dogma;
var
   A, B:          Integer;
   Classes:             Integer;
   Students:            Integer;
   Total, R:      Real;
   Average:       Real;

begin
   Writeln ('Please specify number of classes:');
   Readln (Classes);
   Writeln ('Enter number of students per class:');
   Readln (Students);
   for  A := 1  to  Classes  do
       begin
           Total := 0;
           Writeln ('Class number: ', A);
           for  B := 1  to Students  do
                   begin
                       Writeln ('Enter mark no. ', B, ':');
                       Readln (R);
                       Total := Total + R;
                   end;
```

```
24)             Total := Total / Students;
25)             Writeln ('Average in class ', A, ': ', Total);
26)         end;
27) end.
```

נחקור כעת את התוכנית. בחלק הגדרת המשתנים הגדרנו את המשתנים A ו-B שישמשו אותנו לצורכי ספירה. המשתנים Classes ו-Students הוגדרו לצורכי קליטת מספר הכיתות ומספר התלמידים בכל כיתה, בהתאמה.

בשורה 14 אנו פותחים בלולאה הראשונה שתמנה את הכיתות. כאשר נטפל בכיתה הראשונה, המשתנה A יהיה שווה 1. כאשר נטפל בכיתה השנייה, המשתנה A יהיה שווה 2, וכך הלאה. המשתנה A ימנה מספר פעמים, על-פי הערך שב-Classes (אשר נקלט בתחילת התוכנית). בשורה 16 אנו מאפסים את המונה Total שתפקידו להחזיק את ממוצע הכיתה שבה אנו מטפלים.

בשורה 18 יצרנו לולאה נוספת, שתפקידה לטפל בכל תלמיד. לולאה זו, מונה בעזרת המשתנה B את מספר פעמים על-פי הערך שבמשתנה Students. תוכן לולאה זו כולל הצגת בקשה לציון, קריאת הנתון לתוך משתנה זמני בשם R, והוספתו לערך הכולל Total. כאשר מסתיימת הלולאה, והמחשב מגיע לשורה 24, הוא מחשב את הממוצע של אותה כיתה ומדפיס אותו (שורה 25). כעת חוזר כל התהליך שוב ושוב וכל זאת הודות ללולאה החיצונית, שהמשתנה A משמש בה כמונה.

בדרך זו יצרנו קינון של שתי לולאות. את הטכניקה הזו ניתן ליישם גם לגבי הלולאות while ו- ... repeat ... until. ניתן ליצור קינון עד כמה רמות שנרצה, **אך** עלינו לזכור כלל בסיסי: **אסור** לשנות משתנה (מונה) של לולאה, כלומר, לא ניתן ליצור קינון של שתי לולאות for ולהשתמש במונה אחד. לכל לולאה צריך להיות **מונה נפרד**.

נבחן גירסה נוספת של לולאות מקוננות:

```
program Dogma;
var
   I, Total:      Integer;
   Counter:          Integer;
   Ch:               Char;

begin
   repeat
      Total := 0;
      Counter := 0;
```

```
11)        repeat
12)            Writeln ('Enter next mark:');
13)            Readln (I);
14)            Total := Total  +  I;
15)            Counter := Counter  +  1;
16)        until  I  <  0;
17)        Total := Total  div  Counter;
18)        Writeln ('Average = ', Total);
19)        Writeln ('Do you want to do it again (Y\N)?');
20)        Readln (Ch);
21)    until  Ch  <>  'Y';
22) end.
```

תיאור התוכנית:

תוכנית זו קולטת מספר ציונים ומדפיסה את הממוצע שלהם. כאשר המשתמש רוצה לסיים את קליטת הציונים עבור כיתה מסוימת, הוא מכניס ערך שלילי. לאחר סיום קליטת הציונים וחישוב הממוצע, המחשב שואל את המשתמש אם ברצונו לחזור על התהליך עבור כיתה נוספת.

מהלך התוכנית:

בשורות 3 ו-4 הגדרנו שלושה משתנים שלמים. משתנה כללי (I), משתנה לניהול סכום הציונים (Total) ומשתנה למנייה של מספר הציונים שנקלטו (Counter). בנוסף הגדרנו גם משתנה מסוג תו (Ch) שאת תפקידו נבין בהמשך.

ראשיתה של התוכנית בשורה 8. בשורה זו נמצאת הגדרה לתחילת לולאה מסוג repeat ... until ... המסתיימת בשורה 21. את אופיה ומטרתה של לולאה זו נבין בסוף התוכנית. בשורות 9 ו-10 אנו מתחלים את המשתנים Total ו-Counter לערך 0, כך שבכל פעם שהלולאה repeat ... until ... המוגדרת בין שורות 8 ו-21 תחזור על עצמה, יתוחלו משתנים אלה לערך 0. בנוסף לכך, הגדרנו לולאה נוספת מסוג repeat ... until ... בין השורות 11 ו-16. לולאה זו כוללת הדפסת הודעה למשתמש להכניס את הציון הבא. הציון נקלט לתוך המשתנה השלם I, ומוסף למונה הכללי Total. בנוסף על כך, הורינו למחשב לקדם את המונה Counter ב-1 כדי שיוכל למנות את מספר הציונים שנקלטו. בשורה 16 הצבנו תנאי לולאה: יש לחזור על התהליך עד אשר תוכן המשתנה I קטן מ-0, כלומר עד אשר קלטנו ערך שלילי עבור ציון.

בסיום הלולאה אנו מקבלים שני דברים: את סכום כל הציונים שנקלטו (במשתנה Total) ואת מספר הציונים שנקלטו (במשתנה Counter). לצורך חישוב ממוצע הציונים, אנו מחלקים את הסכום במספר הציונים שנקלטו (שורה 17), ומדפיסים את התוצאה.

עתה, משהסתיים תהליך זה, המחשב יציג הודעה השואלת אם ברצוננו לחזור על התהליך מההתחלה (שורה 19). בשורה 20 אנו קולטים תו בודד שאמור להיות "Y" המייצג את התשובה "כן", או "N" המייצג את התשובה "לא". בשורה 21 הצבנו תנאי: על הלולאה לחזור על עצמה עד אשר התו שבמשתנה Ch אינו "Y", כלומר עד אשר המשתמש מחליט שברצונו לסיים את התוכנית.

תוכנית זו היא דוגמה טובה להדגמה של לולאות מקוננות. אנו רואים שניתן לשלב שתי לולאות שאינן תלויות זו בזו לצורך ביצוע פעולה שאינה קבועה. הפעולה נמשכת על פי הנחיות המשתמש.

בנוסף על כל זאת, ישנם שימושים אחרים ללולאות מקוננות. בדוגמאות שבחנו עד כה הדגמנו כי משתמשים בטכניקה זו כדי לפתור בעיה של מחזוריות בלתי קבועה, זו שתלויה בהחלטת המשתמש. כעת נדגים גם את הצד השני, שבו לא התמקדנו מספיק. לפני שנעשה זאת נבחן את התרגיל הבא: נכתוב תוכנית המציירת מרובע בגודל 4x3.

```
1)  program Dogma;
2)  const
3)     H = 4;
4)     W = 3;
5)
6)  var
7)     A, B:          Integer;
8)  begin
9)     for  A := 1  to  H  do
10)        begin
11)           for  B := 1  to  W  do
12)                  Write ('*');
13)               Writeln;
14)        end;
15) end.
```

בתוכנית זו הגדרנו שני קבועים בשורות 3 ו-4. הקבוע H מגדיר את גובה המרובע (מספר השורות שבמלבן), והקבוע W מגדיר את רוחב המרובע. הגדרנו משתנים אלה למקרה עתידי, בו נרצה לשנות את מימדי המלבן. להגדרות אלו אין כל חשיבות, כי היינו יכולים להציב במהלך התוכנית את הערכים 4 ו-3 על פי הצורך. מכיון שתהליך זה מבהיר את אופן פעולת התוכנית, בחרנו להוסיף קבועים אלה. בשורה 7 הגדרנו שני משתנים, A ו-B, שישמשו מונים.

תחילתה של התוכנית בשורה 9 אשר כוללת לולאת for שתפקידה למנות את מספר השורות היוצרות את המלבן. שורות 11, 12 ו-13 הן תוכן הלולאה וכוללות לולאת for נוספת, שתפקידה ליצור את שורות המלבן. פקודות הלולאה שבשורות 11 ו-12 למעשה "מציירות" שורה בודדת של המלבן. כאשר הלולאה

הראשית (המבוססת על המשתנה A) חוזרת על עצמה 4 פעמים, יודפסו למעשה 4 שורות. תוכנית זו מדגימה היבט נוסף של השימוש בלולאות מקוננות. היא אינה מייצגת כל טכניקה מיוחדת, אלא מראה כיצד אפשר לקצר תהליכים בתוכנית וכך גם "לחסוך" בפקודות. נבהיר זאת. התוכנית צריכה להדפיס מלבן בגודל 4x3, כפי שמודגם בציור הבא:

```
***
***
***
***
```

ניתן לכתוב את התוכנית בצורה הבאה:

```
begin
   Writeln ('***');
   Writeln ('***');
   Writeln ('***');
   Writeln ('***');
end.
```

שש השורות הללו היו נותנות לנו את התוצאות הרצויות. אך, מה היה קורה אילו היה עלינו להציג מלבן ברוחב 5 ובעל 20 שורות? במקרה זה היינו נאלצים להשתמש ב-20 שורות Writeln. לעומת זאת, כאשר משתמשים בתוכנית הקודמת המבוססת על לולאות מקוננות, כל אשר היה עלינו לעשות הוא לשנות את הערכים של הקבועים H ו-W לערכים המבוקשים. אורכה של התוכנית היה נשאר ללא שינוי.

בעזרת הדוגמאות שבחנו עד עתה ניתן להבין כי מטרת הלולאות המקוננות לשמש אותנו לפתרון שתי בעיות:

א. חזרה על קבוצת פקודות לפי פרמטר: לולאות שמספר החזרות שלהן נקבע על ידי המשתמש או על פי תנאים משתנים.

ב. קיצור תהליכים בתוכנית.

מבנה רב-ברירות - CASE

בראשית הפרק למדנו על התניה מסוג **אם...אז...** (**if...then...**). בעבודה שוטפת ניתן להבחין בחיסרון קטן הנובע מעצם מבנה הפקודה. כדי להבין זאת, נתבונן בתרגיל 4.5, שבו אנו מתבקשים "להמיר" ערך מספרי למילה. כלומר, המשתמש מציין ערך שלם ועל התוכנית לכתוב את הערך הזה בטקסט באנגלית. לשם כך עלינו להשוות בין המשתנה שהכיל את הערך הנקלט לבין כל הערכים שעלינו לטפל בהם. קטע התוכנית עשוי להיראות כך:

```
if I  = 1 then Writeln ('ONE');
if I  = 2 then Writeln ('TWO');
```

בתרגיל 4.2 נתבקשתם לבצע בדיקות בהנחה שקיימים רק חמישה מצבים, אך מה היה קורה אילו היינו צריכים לבדוק עשרים מצבים? לצורך פישוט המשימה קיימת "פקודת תנאי" נוספת בשפת פסקל שתפקידה לפשט מצבים אלה:

מבנה CASE - **מבנה רב-ברירות**. בעזרתו אנו יכולים להציג מצבים מורכבים וגם את המטלה שיש לבצע כאשר מתקיים כל אחד מהמצבים האלה. כך נראה מבנה CASE:

```
case [expression] of
   case #n: [statement];
end;
```

עתה נסביר כל מרכיב במבנה זה:

case... of ו-end אלו הן מילים שמורות המגדירות שמשתמשים במבנה רב ברירות.

[expression] הוא הביטוי שאנו רוצים לבחון.

case #n אחד מהערכים שעומד למבחן.

[statement] המטלה שיש לבצע במקרה שמקרה מסוים נבחר.

לדוגמה, נבחן את קטע התוכנית הבא:

```
Readln (I);
case  I  of
  0:  Writeln ('Zero');
  1:  Writeln ('One');
  2:  Writeln ('Two');
  3:  Writeln ('Three');
  4:  Writeln ('Four');
  5:  Writeln ('Five');
end;
```

בקטע תוכנית זו קלטנו ערך שלם לתוך המשתנה I. לאחר מכן יצרנו מבנה רב-ברירות case, שבו הגדרנו את התנאים הללו:

- אם נקלט הערך 0, התוכנית תדפיס את המילה Zero.
- אם נקלט הערך 1, התוכנית תדפיס את המילה One.
- אם נקלט הערך 2, התוכנית תדפיס את המילה Two.
- אם נקלט הערך 3, התוכנית תדפיס את המילה Three.
- אם נקלט הערך 4, התוכנית תדפיס את המילה Four.
- אם נקלט הערך 5, התוכנית תדפיס את המילה Five.

בצורה זו חסכנו את הטירחה בכתיבת שש שורות התניה, אחת לכל מצב. בנוסף, למבנה זה יש תכונות המפשטות בדרגה נוספת את כתיבת התוכנית.

לדוגמה, עלינו לכתוב תוכנית הקולטת ערך מספרי ומדפיסה את המילה FIRST כאשר הערך נע בתחום 1 עד 10, ואת המילה SECOND כאשר הערך נע בתחום 11 עד 20. אם היה עלינו להשתמש בהתניה ...if... then היינו צריכים ליצור שתי התניות (אחת לכל מצב), שכל אחת מהן כוללת ביטוי לוגי מורכב. כאן בא לעזרתנו מבנה רב-הברירות. ניתן יהיה לרשום קטע תוכנית זה באופן הבא:

```
Readln (I);
case I of
    1..10: Writeln ('FIRST');
  11..20: Writeln ('SECOND');
end;
```

הצורה הפשוטה והמובנת של מבנה זה מדגישה את יתרונותיו על ההתניה ...if... then. כבר במבט ראשון ניתן להבין את המשמעות: ההגדרה 1..10 מציינת כי המילה FIRST תודפס כאשר הערכים יהיו בתחום 1 עד 10, בעוד שההגדרה 11..20 מציינת כי המילה SECOND תודפס כאשר הערכים יהיו בתחום של 11 עד 20.

נבחן דוגמה נוספת: עלינו לכתוב קטע תוכנית הקולט ערך שלם בתחום 1 עד 10. התוכנית צריכה להדפיס את המילה Even (זוגי) כאשר הערך המתקבל הוא זוגי, ואת המילה Odd (אי-זוגי) כאשר הערך המתקבל הוא אי-זוגי. נוכל לממש זאת בצורה הבאה:

```
Readln (I);
case  I  of
  2, 4, 6, 8, 10: Writeln ('Even');
  1, 3, 5, 7, 9: Writeln ('Odd');
end;
```



שימו לב: *כאשר מציינים את הערכים שיגרמו להצגת כל הודעה, התו פסיק (,) מפריד בין ערך לערך.*

לפנינו דוגמה נוספת:

כתבו תוכנית הקולטת שני ערכים ממשיים A ו- B ותו בודד המציין אופרטור של סוג פעולת החשבון הרצויה (+ - * /). התוכנית תחשב את התוצאה על פי הנתונים שהוגדרו.

```
1)    program  Dogma;
2)    var
3)        A, B, C:       Real;
4)        Ch:            Char;
5)
6)    begin
7)        Writeln ('Type in two real values:');
8)        Readln (A, B);
9)        Writeln ('Type in the operator (+, -, *, /):');
10)       Readln (Ch);
11)
12)       case  Ch  of
13)           '+':  C := A + B;
14)           '-':  C := A - B;
15)           '*':  C := A * B;
16)           '/':  C := A / B;
17)       end;
18)
19)       Writeln ('Result: ', C);
20)   end.
```

בתוכנית זו הגדרנו שלושה משתנים ממשיים: A ו-B המשמשים לקליטת שני הנתונים, ו-C שיקבל את תוצאת החישוב. בנוסף, הגדרנו גם את המשתנה Ch שיקלוט את האופרטור של פעולת החישוב הרצויה. החלק הראשי של התוכנית הוא מבנה case בשורה 12. כאן אנו מציבים התניות לפעולות החשבון השונות. לדוגמה, התו "+" יגרום לחיבור של ערכי המשתנים A ו-B ולהצבת התוצאה במשתנה C. באותו האופן יתקיימו שאר פעולות החשבון. בסיום התוכנית (שורה 19) תודפס תוצאת החישוב המבוקש.

נעסוק עתה בתכונה חשובה של מבנה CASE. כדי להבין אותה, נחזור לדוגמה הקודמת. בדוגמה זו יכול להיווצר מצב בו המשתמש יבקש לבצע פעולה שאינה קיימת. לדוגמה, במקום להזין תווים לסימול פעולות חיבור, חיסור, כפל או חילוק, הוא עלול ללחוץ בטעות על תו אחר, ואז נרצה להודיע לו על כך. נשתמש לצורך זה בתוספת למבנה CASE שלמדנו, המאפשרת לעשות זאת. תוספת זו היא המילה **else (אחרת)**, שהשימוש בה נעשה כמו בהתניה ...if... then.

נדגים זאת על ידי שיפור שורות 12 עד 17 בתוכנית הדוגמה שלנו:

```
case  Ch  of
   '+': C := A + B;
   '-': C := A - B;
   '*': C := A * B;
   '/': C := A / B;
   else Writeln ('Error: ', Ch);
end;
```

המילה else מנחה את המחשב להדפיס הודעת שגיאה כאשר תתקבל בקשה בלתי חוקית.



הערה: *יש לשים לב כי המילה else חייבת להימצא בסוף כל ההתניות. אם ננסה להציב תנאי זה במקום אחר, המחשב יודיע על שגיאה.*

מגבלות במבנה CASE רב-ברירות

כשם שיש יתרונות למבנה CASE, יש לו גם חסרונות. החיסרון העיקרי הוא בכך שניתן ליישם אותו אך ורק עבור משתנים שלמים ומשתנים מסוג תו, ובאופן כללי - עבור משתנים מסוג "סודר" (מבחינה מתמטית אלו קבוצות בנות-מנייה). אם נשתמש במשתנים לא סודרים, למשל במשתנה ממשי (real), המהדר ידווח על שגיאה.

תרגילים

תרגיל 4.1

כתבו תוכנית, הקולטת את מימדיו של מלבן ומדפיסה אותו בעזרת התו כוכבית.

תרגיל 4.2

כתבו תוכנית הכוללת לולאה שתפקידה לקלוט עשרה נתונים ממשיים. בסופה של הלולאה על המחשב להדפיס את הערך הנקלט בעל השבר הגדול ביותר. לאחר סיום תהליך זה המחשב ישאל את המשתמש אם ברצונו לחזור על התהליך. במידה והמשתמש יאשר זאת, על כל התהליך לחזור מתחילתו.

(**רמז:** יש להשתמש בלולאה ... repeat ... until שתכלול בתוכה את כל תהליך הקלט והבדיקה).

תרגיל 4.3

כתבו תוכנית המשמשת כמשחק בין שני חברים. הרעיון עליו מבוסס המשחק הוא, שעל השחקן הראשון לקבוע ערך ועל חברו לנחש אותו. מהלך התוכנית: המחשב יבקש מן השחקן הראשון להכניס ערך בין 1 ל-100. המחשב יבדוק אם הערך בתחום שנקבע. במידע והערך אינו כנדרש, המחשב יבקש להכניסו שוב, **עד אשר** השחקן הראשון יכניס ערך תקין. לאחר מכן ינקה המחשב את המסך בעזרת שימוש חוזר של הפקודה Writeln (25 פקודות Writeln צריכות לספק ניקוי של המסך).

מעתה יחל השלב השני, שבו על השחקן השני לנחש את המספר. כאשר הוא יכניס ערך גבוה מזה שצוין על ידי השחקן הראשון, המחשב ידפיס את ההודעה "Too high", וכאשר השחקן השני יכניס ערך נמוך מזה שציין השחקן הראשון המחשב ידפיס את ההודעה "Too low". כך יימשך התהליך **עד אשר** השחקן השני ינחש את המספר שנבחר על ידי השחקן הראשון.

תרגיל 4.4

כתבו תוכנית הקולטת שני ערכים שלמים. אם שני הערכים שווים יודפס "Equal", ואם לא - יודפס "Not Equal".

תרגיל 4.5

כתבו תוכנית הקולטת ערך שלם בתחום של 0 עד 4. התוכנית תדפיס את המספר בצורה מילולית. לדוגמה, כשיתקבל הערך 0 יודפס ZERO, כאשר יתקבל הערך 1 יודפס ONE וכך הלאה.

תרגיל 4.6

כתבו תוכנית הקולטת שלושה ערכים שלמים. התוכנית תבדוק אם הערך הראשון נמצא בתחום של שני הערכים הנותרים. הערך הראשון צריך להיות גדול או שווה לערך השני, וגם קטן או שווה לערך השלישי. במקרה כזה, על התוכנית להדפיס OK, במקרה שהתנאי אינו מתקיים על התוכנית להדפיס NOT OK.

תרגיל 4.7

כתבו תוכנית הקולטת שני ערכים שמגדירים גבול עליון וגבול תחתון ומדפיסה את כל הערכים בתחום זה. יש להתחשב בשני מצבים אפשריים: כאשר הגבול התחתון קטן בערכו מהעליון (במקרה זה יש להשתמש במילה to), וכאשר הגבול התחתון גדול בערכו מהגבול העליון (במקרה זה יש להשתמש במילה downto).

תרגיל 4.8

כתבו תוכנית שדומה לתוכנית הקודמת, אך בשינוי מהותי: בתוכנית הקודמת יצאנו מתוך הנחה שיש להשתמש בהתניה כדי לבדוק את היחס בין הגבול התחתון לעליון. את התוכנית הזו יש לכתוב ללא התניה זו.

תרגיל 4.9

כתבו תוכנית הקולטת שני ערכים מהמשתמש ומציגה את כל המספרים בתחום זה. יש לבצע את התהליך בעזרת הלולאה ... repeat ... until.

תרגיל 4.10

כתבו תוכנית הקולטת עשרה ערכים ממשיים וסופרת כמה מהם מכילים שברים עשרוניים.

(**רמז:** יש להשתמש בפונקציה Frac).

תרגיל 4.11

כתבו תוכנית הקולטת ערך ממשי ומדפיסה את ערכו המוחלט ללא שימוש בפונקציה Abs.

תרגיל 4.12

כתבו תוכנית הקולטת שני מספרים ומדפיסה את הגדול מבין שניהם.

תרגיל 4.13

כתבו תוכנית הקולטת מספר שלם ובודקת אם הוא זוגי או לא. התוכנית תדפיס הודעה מתאימה לכל מצב.

(**רמז:** יש להשתמש בשארית של חלוקת המספר ב-2).

תרגיל 4.14

כתבו תוכנית הקולטת תו בודד ובודקת אם הוא אות. אם התשובה חיובית יש להדפיס הודעה מתאימה.

(**רמז:** יש לבדוק אם התו המתקבל הוא גם בתחום "Z".."A" וגם בתחום "z".."a").

תרגיל 4.15

כתבו תוכנית הקולטת שלושה ערכים ממשיים המשמשים מקדמים למשוואה ריבועית. התוכנית תחשב ותדפיס את שורשי המשוואה. אם היא לא תוכל לעשות זאת, היא תדפיס הודעה מתאימה.

תרגיל 4.16

כתבו תוכנית המקבלת את האורכים של שני הניצבים במשולש ישר זווית ומחשבת את היתר על פי משפט פיתגורס.

תרגיל 4.17

כתבו תוכנית הקולטת ערך בתחום 0 עד 180 המייצג זווית. על התוכנית להדפיס את סוג הזווית: ישרה, קהה או חדה.



***הערה**: יש לבדוק שהזווית הנקלטת אכן נמצאת בתחום הזה.*

תרגיל 4.18

כתבו תוכנית הקולטת ערך שלם ומדפיסה אותו מן הסוף להתחלה. לדוגמה, הערך 8512 יודפס כך: 2158.

תרגיל 4.19

כתבו תוכנית הקולטת תו בודד. באמצעות המבנה CASE רב-ברירות על התוכנית להדפיס הודעה מתאימה לפי המקרים הבאים:

א. כאשר התו המתקבל הוא מספר (0..9).

ב. כאשר התו המתקבל הוא אות גדולה (A..Z).

ג. כאשר התו המתקבל הוא אות קטנה (a..z).

ד. כאשר התו המתקבל אינו תואם את אחד המאפיינים שצוינו לעיל.

פרק 5

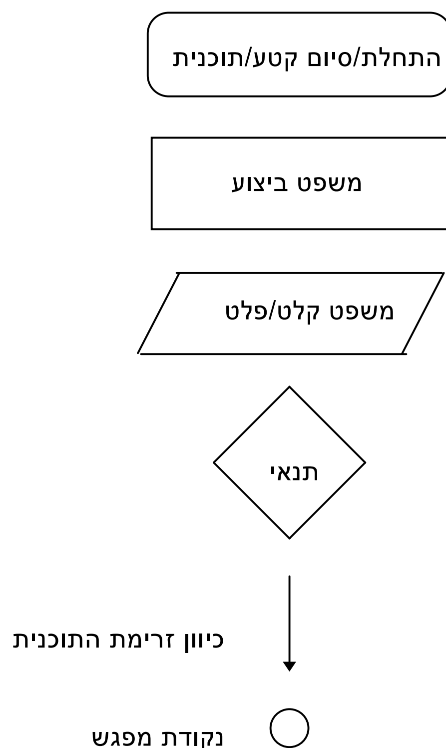
מבני בקרה

הקדמה

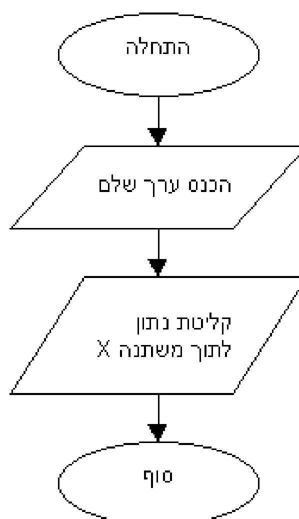
מבני בקרה הם כלי עזר לפיתוח תוכנה. השילוב שלהם מאפשר להציג באופן גרפי תהליכי עיבוד נתונים של מערכת באמצעות **תרשימי זרימה** (**Flow charts**). תרשימים אלה מציגים בפנינו, בצורה ויזואלית, את זרימת המידע, את הפעולות שיש לבצע ואת התנאים לביצוע פעולות שונות כדי להשלים את **משימת עיבוד הנתונים**, ולמעשה הם מתארים את מהלך התוכנית. לרוב, כאשר אנו פונים לכתיבת תוכנית אין אנו יודעים בבירור את מהלכה, את הקטעים שהיא כוללת, סוגי הלולאות, המשתנים שיידרשו לנו, ועוד. תפקיד תרשימי הזרימה הוא לפתור עבורנו את רוב הבעיות הללו. בנוסף, לתרשימי זרימה יש תכונה מיוחדת שהם מתאימים לכל שפה! לא רק זאת, בעזרת תרשים זרימה ניתן לכתוב את התוכנית בדרכים שונות. מכאן, שתרשימי זרימה הם כלי הכרחי לפיתוח תוכנות.

כללים בסיסיים ליצירת תרשימי זרימה

כדי ליצור אחידות והבנה, נקבעו סימנים מוסכמים שבעזרתם יוצרים תרשימי זרימה.



באמצעות סימנים אלה ניתן לתאר באופן כללי את מהלך התוכנית. לדוגמה, מוטל עלינו להציג הודעה המבקשת מן המשתמש להזין למחשב ערך שלם כלשהו ולקלוט אותו למשתנה X. נתאר תוכנית זו בתרשים זרימה:



בצורה זו תארנו את מהלך התוכנית. כדי ללמוד יותר על מבני בקרה אלה ולפתח תוכניות מורכבות יותר, נרחיב את הדיון בנושאים אחדים (ראה בסעיפים הבאים).

סוגים שונים של מבני תכנות

מבני בקרה כוללים שלושה סוגים עיקריים:

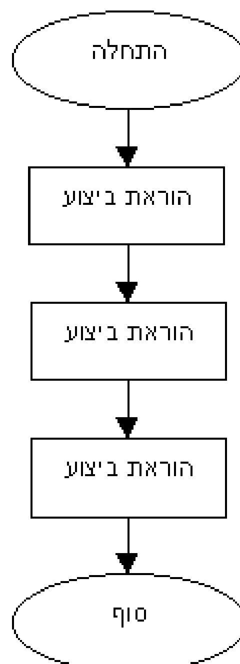
א. מבנה סדרתי.

ב. מבנה תנאי.

ג. מבנה לולאה.

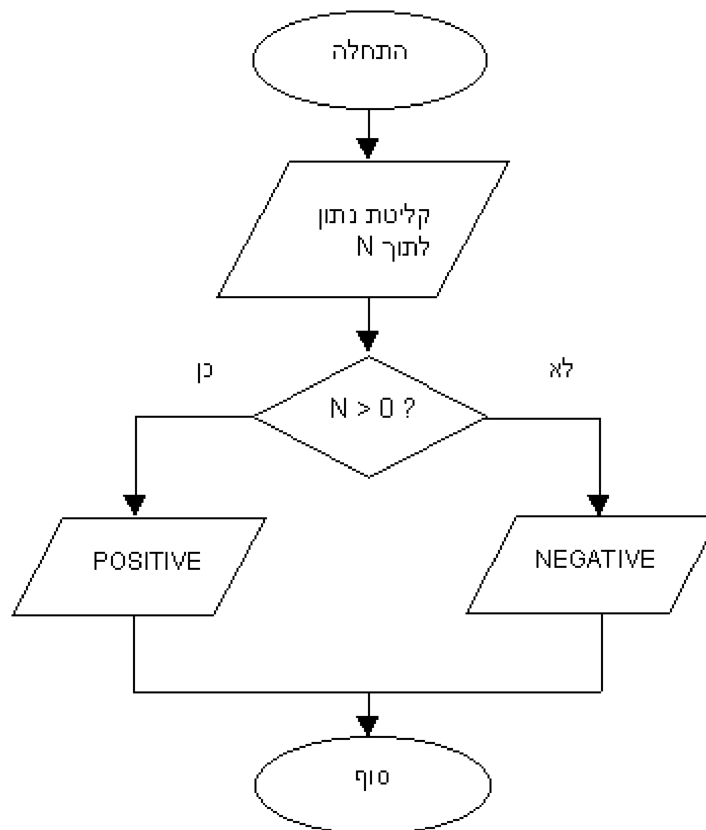
מבנה סדרתי

מבנה מסוג זה הוא הפשוט ביותר. התרשים שזה עתה הצגנו הוא דוגמה טובה למבנה סדרתי. תרשים זה מתאר מהלך סדרתי של תוכנית, כלומר סדר פעולות בצורה קבועה. אין כל התניות בדרך, אין כל קפיצות לנקודות שונות בתרשים, אלא תיאור סדרת פעולות רצופה.



מבנה תנאי

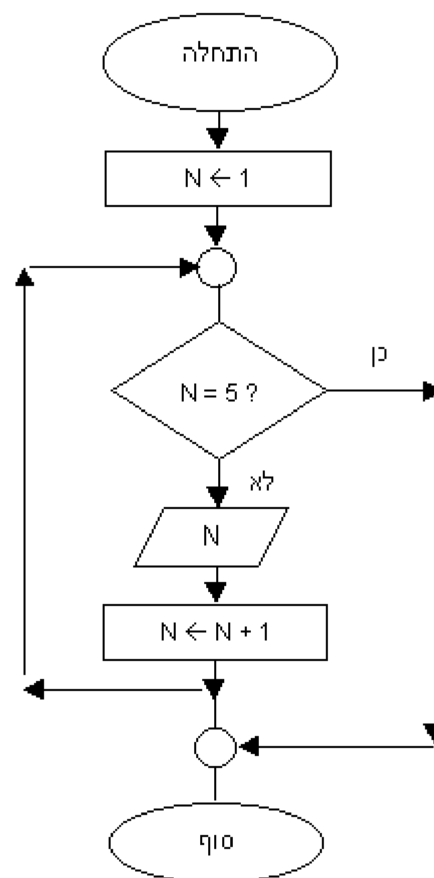
מבנה זה הוא הנפוץ ביותר, מכיון שרוב התוכניות במחשב משתמשות בתנאים לפעולה או בלולאות. במבנה זה אנו שואלים בדרך כלל שאלה, ופועלים על פי התוצאה. נבחן את הדוגמה הבאה: עלינו לכתוב קטע של תוכנית הקולט ערך שלם. אם הערך הוא חיובי, על התוכנית להדפיס את ההודעה POSITIVE, ואם הערך שלילי, על התוכנית להדפיס NEGATIVE. נשרטט תרשים זה:



בתרשים זה תיארנו את אופן הפעולה של תוכנית הקולטת נתון לתוך המשתנה N ולאחר מכן מציבה תנאי. התנאי הנשאל הוא: האם ערכו של המשתנה N גדול מ-0 (N > 0)? במקרה שהתשובה חיובית, התרשים מפנה אותנו לכיוון הוראת פלט המדפיסה POSITIVE. במקרה והתנאי לא מתקיים, התרשים מפנה אותנו לכיוון הוראת פלט המדפיסה את ההודעה NEGATIVE. לאחר ביצוע אחת מההוראות הללו, אנו מופנים להמשך התוכנית. במקרה שלנו מסתיימת התוכנית בשלב זה, אך במקום זאת היינו יכולים להוסיף פעולות שונות.

עתה, משבחרנו תרשים זרימה מורכב יותר, נוכל לדון באחד המאפיינים החשובים של תרשימי הזרימה: **חופש בחירה.** אם היינו מביטים בתרשים שהצגנו בפעם הראשונה, היינו בוחרים בהתניה ...if... then כדי לייצג את התנאי, אך אין זה תמיד כך. מכיון שאין אנו מציינים בבירור צורות התניה מסוימות או מבנה כלשהו, נשאר לנו חופש הבחירה להחליט כיצד ליצור את התהליך.

נבחן דוגמה נוספת שתאפשר להמחיש מאפיין זה: עלינו לכתוב קטע של תוכנית המציג את המספרים מ-1 עד 5. נשרטט את התרשים.

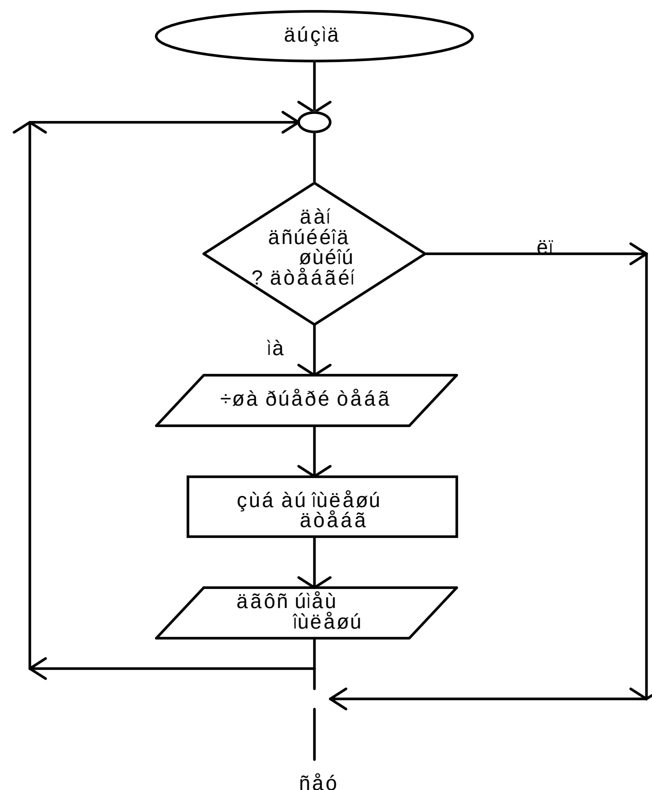


בתרשים זה מתוארת לולאה שתפקידה להציג את הערכים 1 עד 5. כאן אנו יכולים להגיד בבירור כי חופש הבחירה שמקנה לנו התרשים מאפשר לבחור בכל לולאה שהיא, מתוך אלו שהכרנו:
for.. do.. ,while... או repeat ... until ....

מבנה לולאה

הלולאה, כפי שראינו, היא הרחבה של ההתניה. ההבדל המשמעותי שניתן להבחין בו הוא, שההתניה תוביל לבסוף לנקודת מוצא מוגדרת מראש. ללא תלות במסלול שנבחר, תמיד המוצא יהיה בנקודה מסוימת מוגדרת. הלולאה, לעומת זאת, תבצע תנאי בודד ותבחר במסלול שיחזור על עצמו מספר פעמים עד אשר יהא עליה לסיים את פעולתה. בשלב זה היא תבחר במסלול השני שיוביל אותה לנקודת המוצא.

נבחן מבנה נוסף של לולאה:



בלולאה זו אנו שואלים שאלה: "האם הסתיימה רשימת העובדים?" אין זו סתם שאלה, כי כאשר אנו יוצרים תרשים במבנה זה, אין אנו יודעים מראש כמה עובדים יש, מתי להמשיך ומתי לסיים את הפעולה. כלומר, אין אנו מתחייבים על כיוון פעולה מסוים. אנו יכולים לזהות את סיום רשימת העובדים על ידי מונה, שם אחרון ברשימה, סימון כלשהו, או בעזרת קלט מבחוץ.

את השאלה הזו אנו יכולים לפרש בכל דרך שתעזור לנו לפתור את הבעיה. בנוסף, את סוג הלולאה נוכל גם כן לבחור על פי ראות עינינו.

לאחר בירור תנאי זה, אנו יכולים לפנות לשני נתיבים: לנתיב הפעולה של טיפול בנתוני העובד, או לכיוון נקודת המוצא של הלולאה כאשר מסתיימת רשימת העובדים.



זכרו: *לכל מבנה לולאה או תנאי יש נקודת כניסה אחת ונקודת מוצא אחת.*

במקרה והרשימה לא הסתיימה, אנו פונים לנתיב עיבוד הנתונים. בנתיב זה אנו קוראים תחילה את נתוני העובד. גם כאן אין אנו מציינים בפירוש את הנתונים שברצוננו לקרוא. נוכל, לדוגמה, לקרוא סוגים שונים של נתונים מכיון שהנתונים של כל העובדים אינם שווים (יש הבדלים במצב משפחתי, מספר הילדים, תשלומים על חשבון הלוואה, תרומות וכדומה). לאחר קריאת הנתונים אנו מחשבים את המשכורת. בתהליך זה אנו יכולים לשקלל את הוותק, הגיל, סוג העבודה או כל מאפיין אחר שנבחר. בסוף נתיב זה אנו מדפיסים את תלוש המשכורת וחוזרים לנקודת הכניסה של הלולאה. שוב חוזר על עצמו התהליך, עד אשר מסתיימת הרשימה.

אלגוריתם (Algorithm)

אלגוריתם הוא מושג המתאר סדרת פעולות חד-משמעיות, ניתנות לביצוע, אשר הוצאתם לפועל תוביל בוודאות לפתרון תוך מספר סופי של צעדים. האלגוריתם מציג את הדרך להשגת המטרה שלשמה הוגדר התהליך.

ברצוננו לנסח פתרונות לבעיות אשר יציגו את דרך הפתרון לפרטיה, אך מבלי להציג את הפתרון בשפת תכנות כלשהי. היתרון בניסוח שאינו תלוי בשפה הוא שניתן לממש אותו בכל שפת תכנות. בניסוח הפתרון המילולי אנו מתמודדים עם כל המטלות שמציבה הבעיה בפנינו, מבלי לעסוק "בבעיות טכניות". הפתרון המילולי מקביל במידה רבה לפתרון באמצעות תרשים זרימה. שתי הדרכים מאפשרות יצוג אלגוריתם לבעיה ללא תלות בשפת התכנות.

הניסוח של אלגוריתם מילולי חייב לענות על תכונות הבסיס של אלגוריתם, כפי שתארנו אותו.

"אלגוריתם הוא סדרת הוראות". התכונה הזו באה לידי ביטוי במבני הבקרה הקיימים. נרשום את מבני הבקרה הבסיסיים בניסוח מילולי מוסכם.

מבנה סדרתי

התחל
הוראה 1
הוראה 2
....
הוראה I
....
הוראה n
סוף

מבנה תנאי מצומצם

אם < תנאי = אמת > **אזי**
התחל
הוראה 1
הוראה 2
....
הוראה I
....
הוראה n
סוף

מבנה תנאי מורחב

אם < תנאי = אמת > **אזי**
התחל
הוראה 1
הוראה 2
....
....
הוראה I
....
הוראה n
סוף, אחרת
התחל
הוראה 1
הוראה 2
....
הוראה I
....
הוראה n
סוף

מבנה לולאת while

כל עוד < תנאי = אמת > **בצע**
התחל
הוראה 1
הוראה 2
....
הוראה I
....
הוראה n
סוף

מבנה לולאת repeat

חזור
הוראה 1
הוראה 2
....
הוראה I
....
הוראה n
עד < תנאי = אמת >

מבנה רב-ברירות - CASE

בצע לפי ערך < ביטוי>
ערך 1: משפט ביצוע 1
ערך 2: משפט ביצוע 2
....
ערך n: משפט ביצוע n
סוף

"הוראות חד-משמעיות הניתנות לביצוע"

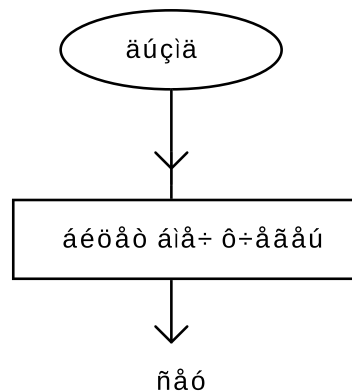
כאשר משתמשים בניסוח מילולי לאלגוריתם רצוי שנקפיד על תכונה זו. נשתמש במילות ציווי כפי שנדרש, כדי לקבל קבוצת מילים המאפשרת כתיבה וניסוח של אלגוריתם מילולי.

הנה לדוגמה מספר מילות ציווי ופירושיהן :

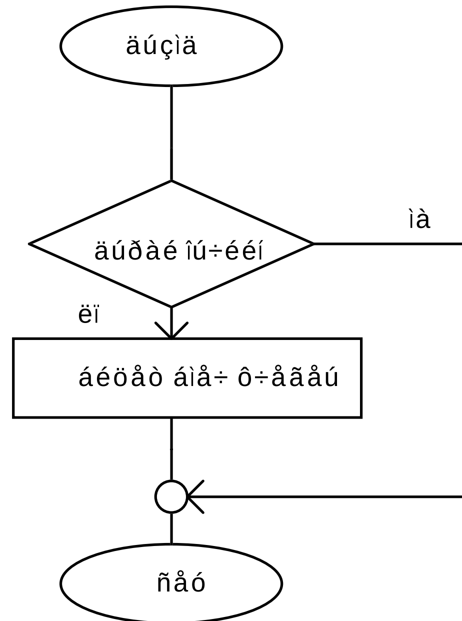
המילה	**הסבר**
חשב	חישוב ביטוי כלשהו והשמתו במשתנה
קלוט	קליטת ערכים למשתנה
הדפס	הדפסת ערכים לפלט

את מבני התכנות שהצגנו קודם כאלגוריתמים, נוכל להציג גם באמצעות תרשימי זרימה. הבה נעשה זאת.

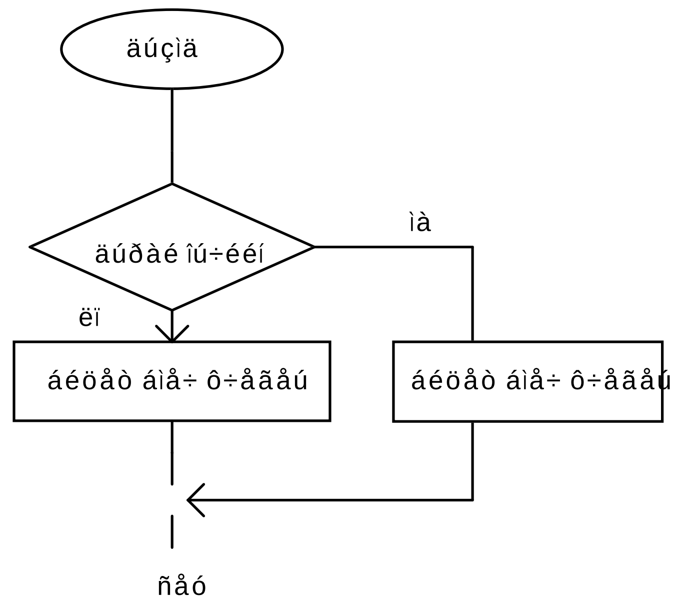
מבנה סדרתי



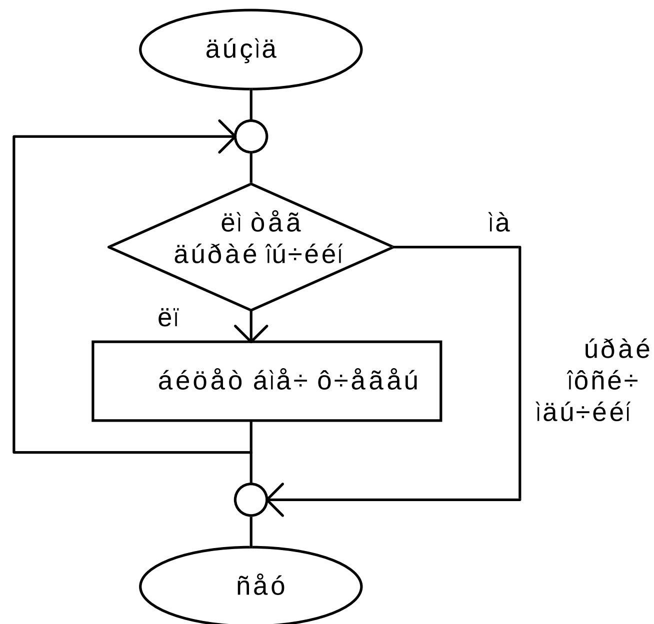
מבנה תנאי מצומצם



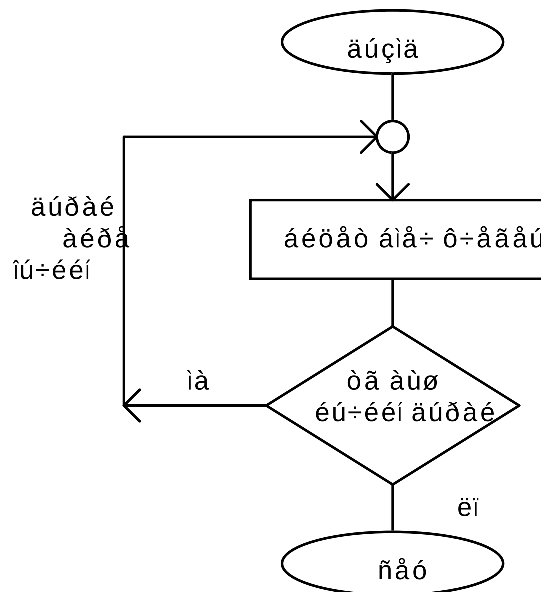
מבנה תנאי מורחב



מבנה לולאת WHILE



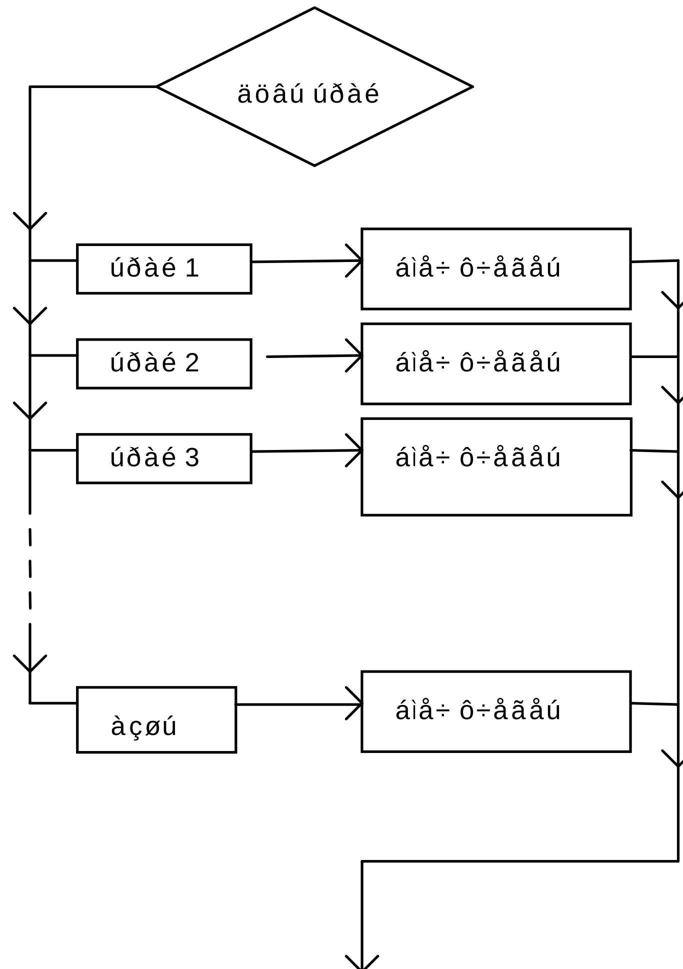
מבנה לולאת REPEAT



כאן אנו רואים כי מנגנון הבקרה נמצא בסוף הלולאה. מצב זה מאפשר לשלוט על התהליך על פי התוצאה של פעולה מסוימת. לדוגמה, נכתוב תוכנית אשר שולטת על פעולת מכונה. הלולאה שבה מדובר צריכה להתקיים **עד אשר** יתברר כי המכונה הפסיקה לפעול. במקרה אחר נקלוט נתונים ונדפיס אותם עד אשר נקבל הוראה המציינת שיש להפסיק את התהליך.

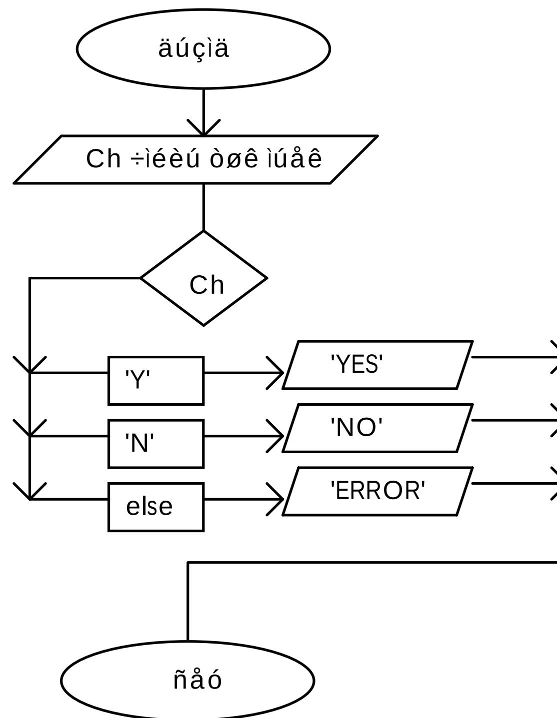
מבנה רב-ברירות -CASE

בפרק 4 למדנו על מבנה מיוחד: מבנה רב-ברירות. מבנה זה פתר אותנו משימוש במספר רב של התניות. בשל מבנהו המיוחד, גם אופן תיאורו הסכימתי שונה משאר המבנים שלמדנו. במבנה הכללי, יחידה זו כוללת את התנאי, ומשם מתחילות הסתעפויות, כך שכל הסתעפות מייצגת מצב אחר. נבחן את מבנה ההתניה הזה:



כאן אנו רואים כי בראש המבנה ניצבת התניה או משתנה. מהתניה זו מסתעפים תנאים או ערכים, שכאשר אחד מהם נבחר, מבוצע בלוק הפקודות שקשור אליו.

לדוגמה, כאשר נקלוט מקש בודד ונרצה לבחון אם נלחץ "Y" או "N" ניצור תרשים כזה:



כך נראה תרשים המגדיר קטע תוכנית שתפקידה לקרוא תו בודד ולבחון אותו. אם נקלט "Y" תציג התוכנית את המילה "YES", ואם נקלט "N" היא תדפיס "NO". בכל מקרה אחר תודפס המילה "Error".

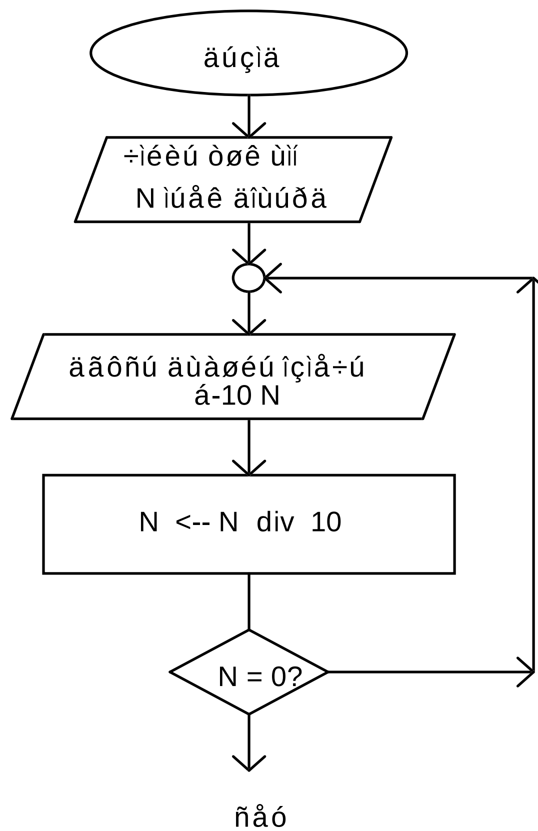
תרגילים

תרגיל 5.1

כתבו תרשים זרימה ותוכנית לקליטת שלושה ערכים שלמים. התוכנית תמיין את שלושת הערכים ותציג אותם מהגדול לקטן.

תרגיל 5.2

נתון תרשים הזרימה הבא:



כתבו תוכנית המבצעת תהליך זה.



הערה *: מבנה הלולאה יכול לרמוז על הסוג המומלץ לשימוש.*

תרגיל 5.3

שרטטו תרשים זרימה של תוכנית הקולטת מאה ערכים ממשיים. על התוכנית להדפיס את הנתונים הבאים:

א. את מספר הנתונים שכללו שבר עשרוני.

ב. מספר הנתונים שכללו ערך (עם או בלי שבר עשרוני)

ג. מספר הערכים שהיו שווים ל-0.

תרגיל 5.4

כתבו תרשים זרימה ותוכנית כדי לקלוט עשרה ערכים ממשיים. על התוכנית להדפיס את סכום השברים העשרוניים.

תרגיל 5.5

כתבו תרשים זרימה ותוכנית לקליטת שני מספרים שלמים וחישוב מכפלתם **ללא** שימוש באופרטור הכפל.

(**רמז**: כפל הוא למעשה תהליך חוזר של חיבור).

תרגיל 5.6

כתבו תרשים זרימה ותוכנית לקליטת ערכים ממשיים. התוכנית תחשב את סכום הערכים שנקלטו. הערכים ייקלטו בלולאה שתסתיים כאשר הערך הנקלט יהיה 0.

תרגיל 5.7

כתבו תרשים זרימה ותוכנית לקליטת "זרם" של תווים באנגלית. התוכנית תמשיך לקלוט תווים עד אשר תתקבל נקודה (.). על התוכנית למנות ולהציג את מספר הפעמים שנקלטה האות "A".

תרגיל 5.8

כתבו תרשים זרימה בלבד לתיאור הפעולות להדפסת לוח הכפל.

(**רמז**: יש להשתמש בלולאות מקוננות).

פרק 6

פרוצדורות ופונקציות

הקדמה

בפרק 2 הסברנו שרוב הפקודות בשפת פסקל מתחלקות לשני סוגים: פרוצדורות ופונקציות. גם ראינו כי הפרדת הפקודות לשני סוגים עיקריים אלה מקנה לנו כלים חזקים לפיתוח. בפרק זה נלמד כיצד יוצרים פרוצדורות ופונקציות כדי לנצל את היתרון במבני תכנות אלה. בנוסף, תוך הסתמכות על כלי תכנות אלה, נלמד על מושג המודולריות והעוצמה שהיא מקנה לנו.

מושגי יסוד בהגדרת תת-שגרות (Subroutines)

כדי שנוכל לגשת להגדרת הפרוצדורות, דרושים לנו כמה כלים בסיסיים להבנת מבנה התכנות הזה. כאשר אנו דנים בקטע תוכנית, אנו נוטים לקרוא לה **שיגרה** (**Routine**), מכיון שהקטע מייצג את מהלך התוכנית. כשאנו קוראים מתוך שיגרה זו לפרוצדורה או פונקציה, אנו למעשה מבצעים "סיעוף" מתוך רצף פקודות התוכנית. מכיון שלתוכנית הראשית אנו קוראים **שיגרה** (**Routine**), הרי שלקטע התוכנית הזה נקרא **תת-שיגרה** (**Subroutine**). כבר כאן נסביר כי אין כאן כל כוונה ליצור פקודות חדשות, אלא להפוך את מבנה התוכנית למודולרי וגם לקצר ולקצץ במספר הפקודות. כדי להבין את שתי המטרות הללו נאלץ לפנות להגדרת הפרוצדורות והפונקציות.

תת-שיגרה בסיסית (Basic type subroutine)

המבנה הבסיסי של הפרוצדורה כולל הגדרת שם ואוסף פקודות. תחילה נפנה ללמוד את אופן הגדרת שם תת-השיגרה:

```
procedure ProcName;
```

procedure היא מילה שמורה, אשר גורמת למחשב להבין כי ברצוננו ליצור תת-שיגרה חדשה. שם תת-השיגרה הוא ProcName. לדוגמה:

```
procedure ReadNames;
```

שורה זו מגדירה למחשב תת-שיגרה חדשה בשם ReadNames.

ומה בדבר גוף תת-השיגרה? גוף תת-השיגרה מוגדר בבלוק פקודות: begin...end;. התכונות של בלוק הפקודות בתת-השיגרה זהות לאלו של תת-בלוק שלמדנו במבנים של לולאות והתניות.

נתבונן בתוכנית הדוגמה הבאה:

```
1)  program Example;
2)  var
3)    Age:        Integer;
4)
5)  procedure ReadAge;
6)  begin
7)    Writeln ('Enter your age:');
8)    Readln (Age);
9)  end;
10)
11) begin
12)   ReadAge;
13)   Writeln ('Your age is ', Age);
14) end.
```

נחקור את מהלך התוכנית.

בחלק ההגדרה של התוכנית (שורה 3) יצרנו משתנה בשם Age מסוג שלם. התוכנית מתחילה בשורה 12 ומסיימת בשורה 13, כלומר - שתי שורות בלבד. השורה הראשונה של התוכנית קוראת לתת-שיגרה החדשה שיצרנו. בעשותנו זאת, אנו מפנים את המחשב לשורה 7, לתחילת תת-השיגרה. בשורה זו אנו מדפיסים הודעה המבקשת מן המשתמש להכניס את גילו. בשורה שלאחריה (שורה 8) אנו קולטים את הגיל לתוך המשתנה Age. בסיום תהליך זה (שורה 9) המחשב חוזר לרצף הפקודות הקודם (שורה 13). כך, למעשה, ניתן לחלק את התוכנית לחלקים, או במילים אחרות - ליצור מודולריות.

בתהליך המודולציה אנו "מפרקים" את התוכנית למספר חלקים שכל אחד מהם אחראי לביצוע פעולה אחת כלשהי. זהו נושא חשוב מאוד, מפני שעל ידי חלוקת התוכנית לקטעים ניתן להשיג מספר דברים:

א. התוכנית פשוטה וקלה להבנה.

ב. התוכנית נוחה לתחזוקה - הכנסת שינויים ואיתור ותיקון תקלות (תפקיד כל קטע תוכנית מוגדר מראש וברור).

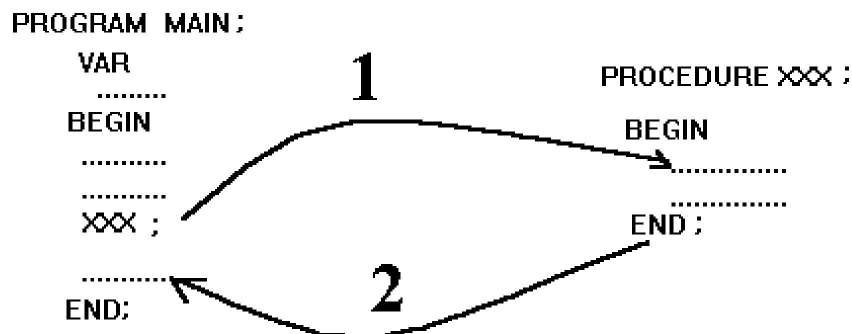
ג. צמצום התוכנית על-ידי שימוש חוזר בתת-שיגרה.

ד. הרחבת פקודות השפה.

על פי חתכים אלה אנו רואים כי תת-השגרות נותנות יכולת לנהל את התוכנית בצורה טובה יותר. בפרק זה נדון בעיקר בחתך השלישי והחתך הרביעי. חתכים אלה הן שתי הסיבות העיקריות לשימוש בתת-שגרות.

תהליך הביצוע של תת-שיגרה

כאשר ניתחנו את מהלך התוכנית הסברנו כי בשורה הראשונה שלה (שורה 12) היא מפנה את המחשב אל תת-השיגרה ReadAge, כך שמהלך התוכנית "מופנה" לשורה 7. התרשים הבא מתאר את אופן הטיפול של המחשב בתת-שגרות:



תרשים זה מתאר תוכנית אשר במהלכה היא קוראת לתת-שיגרה XXX. כאשר התוכנית מגיעה אל הפקודה להפעלת תת-השיגרה, המחשב בוחר במסלול 1 ומפנה את מהלך התוכנית לתחילת קטע הפקודות של תת-שיגרה XXX. בסיום תת-השיגרה, המחשב משתמש במסלול 2 ופונה להמשך התוכנית המרכזית, כלומר לשורה הבאה.

העברת פרמטרים (נתונים) לתת-שיגרה

אחת התכונות החשובות של תת-שיגרה היא היכולת שלה לקבל נתונים. כדי להבין מושג זה נתבונן בפקודה כמו Writeln, המקבלת את הנתונים שעליה להדפיס ולבצע את המוטל עליה. אם לא היינו מספקים לה נתונים, היא לא היתה יכולה לפעול. לרוב, כאשר כותבים תת-שיגרה למטרה מסוימת, נאלצים לפעול על פי "בקשה חיצונית", כלומר על פי נתונים שסופקו מבחוץ. אם כן, אופן ההגדרה של תת-השיגרה, כפי שלמדנו, לוקה בחסר. כדי להשלים חלק חסר זה נלמד את תבנית ההגדרה המורחבת של תת-השיגרה:

```
procedure ProcName (VarName : VarType);
```

בתבנית הגדרה זו אנו מורים למחשב שברצוננו ליצור תת-שיגרה בשם כלשהו, הדורשת נתונים. **VarName** מייצג את שם המשתנה שינהל את הנתון ו-**VarType** הוא סוג הנתון. דוגמה:

```
procedure WriteInteger (I : Integer);
begin
   Writeln ('Value of I: ', I);
end;
```

כעת הגדרנו תת-שיגרה בשם WriteInteger שדורשת ערך שלם כלשהו, ואחר כך היא מדפיסה אותו הערך עם הודעה מתאימה. אם כן, הגדרת משתנה לצורך העברת נתונים לתוך תת-שיגרה מתבצע בדומה להגדרת נתונים בחלק ההצהרה של התוכנית.

תוכנית דוגמה:

```
program Example;
var
   I, N1, N2:     Integer;

procedure Draw (I : Integer);
begin
   for  N2 := 1  to  I  do
      Write ('*');
   Writeln;
end;

begin
   Writeln ('Enter a limit:');
   Readln (I);
   for  N2 := 1  to  I  do
      Draw (N2);
end.
```

נחקור את מהלכה של התוכנית:

בחלק ההגדרה של התוכנית יצרנו שלושה משתנים שלמים. בתחילתה של התוכנית (שורה 13) אנו מציגים הודעה המבקשת מהמשתמש להכניס את הגבול, ולאחר מכן קוראים ערך זה לתוך משתנה (שורה 14). בשורה שלאחר מכן אנו יוצרים לולאה הקוראת לתת-שיגרה Draw פעם אחת בכל מחזור. כדי לנתח את התוצאות נבחן את תת-השיגרה ולאחר מכן נחזור לתוצאות הלולאה.

תת-השיגרה כוללת לולאת for שמקבלת ערך מסוים מבחוץ ועל פיו היא מדפיסה מספר מסוים של כוכביות. לדוגמה, אם הערך שהתקבל מבחוץ הוא 5, אזי יודפסו חמש כוכביות בשורה. תת-השיגרה כוללת שורה נוספת (שורה 9) שתפקידה לרדת לשורה הבאה בתור. כעת נוכל לחזור ללולאה שבשורה 15 ולנתח את התוצאות הסופיות. בתחילה, כאשר N2 מקבל את הערך 1 וקורא לתת-שיגרה Draw, על המסך מופיעה כוכבית בודדת. לאחר מכן, כאשר ערכו של המשתנה N2 מתקדם לערך 2 תת-שיגרה Draw תדפיס שתי כוכביות, וכך הלאה. אם כן, התוכנית תצייר משולש ישר זווית על המסך.

נתבונן בפלט דוגמה זה:

```
Enter a limit:
5
*
**
***
****
*****
```

מכיון שבחרנו שרירותית את הערך 5, יודפסו חמש שורות בלבד על המסך.

משתנים כלליים (Global variables) ומשתנים מקומיים (Local variables)

עתה נבחן את שורה 5, הכוללת את הגדרת תת-השיגרה. בשורה זו הגדרנו משתנה I מסוג שלם. גם בשורה 3 הגדרנו משתנה בשם I מסוג שלם. מדוע המחשב לא הודיע על בעיה? מדוע התוכנית רצה ללא כל תקלות? לשאלות אלו יש פתרון מיוחד.

בשפת פסקל ניתן להגדיר **משתנים כלליים** (**Global variables**) אשר מוגדרים בחלק ההגדרה של התוכנית (תחת המילה השמורה var) ונגישים מכל אזור בתוכנית. כל חלק של התוכנית בכל מקום יכול להשתמש בהם ללא הגבלה. בנוסף לכך, ניתן להגדיר גם **משתנים מקומיים** (**Local variables**) אשר נגישים אך ורק על ידי תת-השיגרה שבה הם מוגדרים.

בזמן שבו תת-השיגרה פעילה, המחשב יקצה זיכרון לניהול משתנים אלה, עד אשר תסיים תת-השיגרה את פעולתה (ואז משתנים אלה יימחקו מהזיכרון). כאשר הגדרנו את המשתנה I בפעם השנייה (שורה 5 - הגדרת תת-השיגרה) המחשב הבין כי אנו רוצים ליצור משתנה מקומי לצורכי תת-השיגרה בלבד. למעשה, כל משתנה שמוגדר לצורך העברת נתונים לתת-שיגרה הוא משתנה מקומי. בנוסף, ניתן לומר כי למשתנים המקומיים יש עדיפות על פני משתנים כלליים, כלומר, כאשר יהיו שני משתנים באותו שם (אפילו תחת סוג הגדרה אחר) יעדיף המחשב להשתמש במשתנים המקומיים בזמן הרצת תת-השיגרה. כאשר נרצה להגדיר משתנים מקומיים נוספים, נוכל לעשות זאת בעזרת הגדרה עם המילה השמורה var:

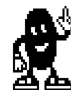
```
program Example
var
   X, Y:        Integer;

procedure Dugma;
var
   X, Y:        Integer;

begin
  .
  .
end;

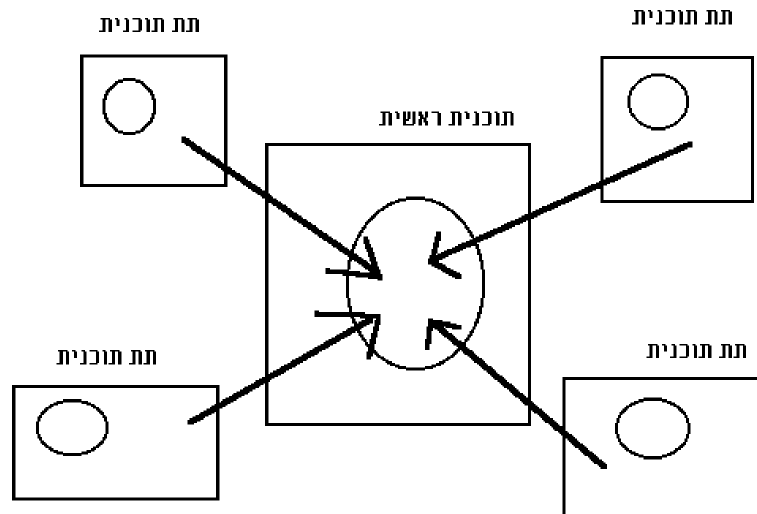
begin
  .
  .
end;
```

בדוגמה זו הגדרנו פעמיים את המשתנים X ו-Y. כאשר נשתמש בהם מחוץ לתת-השיגרה Dugma, המחשב יעשה שימוש באלה שהוגדרו בראש התוכנית. כאשר המחשב ייכנס לתת-שיגרה, הוא יפנה מקום בזיכרון עבור שני משתנים חדשים, כך שתת-השיגרה תוכל להשתמש בהם כבמשתנים מקומיים. כמובן, כאשר תסיים תת-השיגרה את פעולתה, המחשב ימחק את שני המשתנים המקומיים מן הזיכרון, ויאפשר לתוכנית לחזור להשתמש במשתנים הגלובליים.



הערה: *שינוי תוכן המשתנים המקומיים X -Y בזמן ריצת תת-השיגרה אינו משנה את תוכן התאים של X -Y שהוגדרו כמשתנים גלובליים.*

לצורך הבנת הרעיון, נתבונן בתרשים הבא:



כל מלבן מייצג קטע של תוכנית. המלבן הגדול הוא השיגרה הראשית של התוכנית, ואילו המלבנים הקטנים הם תת-שגרות. בנוסף, העיגולים מייצגים משתנים, כך שהעיגול המרכזי מייצג משתנים גלובליים (שהוגדרו בראשית התוכנית) והעיגולים הקטנים מייצגים משתנים מקומיים השייכים כל אחד לתת-השיגרה בה הוא נמצא. על פי תרשים זה ניתן לראות כי כל תת-שיגרה תוחמת בתוכה משתנים מקומיים, ובנוסף היא יכולה להשתמש במשתנים גלובליים. על ידי הבחנה נוספת אנו יכולים להבין גם, כי לתוכנית הראשית אין גישה למשתנים המקומיים של כל תת-שיגרה, ולכל תת-שיגרה אין גישה למתחם המשתנים של תת-שיגרה אחרת.



שאלה: מתי רצוי להגדיר משתנים מקומיים, ומתי רצוי להגדיר משתנים כללים. מהם היתרונות והחסרונות לכל שיטה?

תשובה: לא ניתן להגדיר מצב מדויק בו נעדיף להגדיר משתנים מקומיים על גלובליים, או ההיפך. למרות זאת, ניתן לומר כי לעיתים, כאשר נרצה להשתמש במשתנים לחישובים זמניים או לניהול לולאה, רצוי להגדיר משתנים מקומיים. למרות זאת, כאשר נרצה להשתמש במשתנים ולשמור את ערכם לאחר ביצוע תת-השיגרה, עלינו להגדיר משתנים גלובליים, מכיון שמשתנים מקומיים יימחקו לאחר סיום תת-השיגרה. דרך נוספת היא להעביר את המשתנים כפרמטרים לפי כתובת, כפי שנלמד בהמשך.

הגדרת מספר פרמטרים

כאשר נרצה להגדיר, לדוגמה, שלושה ערכים של תת-שיגרה מסוג שלמים, נוכל לעשות זאת כך:

```
procedure ProcName (A, B, C : Integer);
```

כשם שמגדירים משתנים תחת המילה השמורה var, כך אנו יכולים להגדיר משתנים להעברת נתונים אל תוך תת-השיגרה. נתבונן במקרה נוסף: ברצוננו להגדיר תת-שיגרה שמקבלת שלושה סוגי נתונים: ממשי, שלם ובוליאני. איך נוכל להגדיר זאת עכשיו? כדי להבין את התשובה, נתבונן בדוגמה הבאה:

```
procedure ProcName (R : Real; I : Integer; B : Boolean);
```

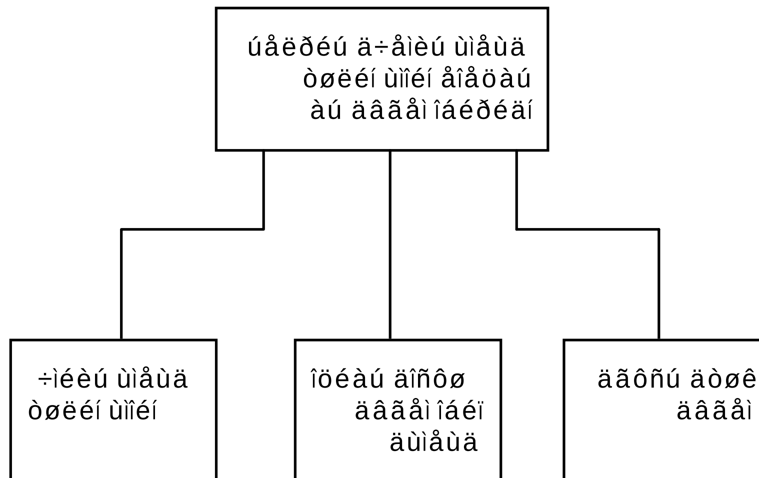
אנו רואים כי כאשר צריך להשתמש בכמה סוגי משתנים, אנו חוצצים בין הגדרותיהם בעזרת התו נקודה-פסיק (**;**).

תהליך פירוק תוכנית במבנה "מעלה-מטה" (Top-down)

כאשר דנו ביתרונות תכנות מודולרי ראינו ששיטת תכנות זו מפשטת את מבנה התוכנית. במציאות, כאשר ארגון פונה לפיתוח תוכנה, מבצעים זאת לרוב מספר מתכנתים, במקביל. כל מתכנת כותב חלק מסוים בתוכנית. נגדיר תחילה את מטרת התוכנית:

úåëðéú ä÷åìèú ùìåùä òøëéí ùìîéí åîåöàú àú äâãåì îáéðéäí

כעת, משהגדרנו את מסגרת התוכנית, נגדיר את שאר מרכיביה:



זהו **תרשים Top-down**. הוא נקרא כך מפני ש**החלק העליון** (**Top**) הוא מטרת התוכנית ו**החלק התחתון** (**Down**) הוא הפירוק לחלקים. יש לציין כי אין זה תרשים זרימה, אלא פירוט של חלקי התוכנית.

במצב שכזה, לאחר שתכננו את חלקיה השונים של התוכנית, נוכל לגשת לאופן כתיבתה. ראשית נפנה להגדרת המשתנים בתוכנית:

```
program Example;
var
   A, B, C:      Integer;
   Largest:     Integer;
```

בקטע זה הגדרנו ארבעה משתנים מסוג שלמים. המשתנים A, B ו-C יקלטו לתוכם שלושה ערכים שלמים שהמשתמש יזין, ואילו המשתנה Largest יכיל בסופה של התוכנית את הערך הגדול מביניהם.

עתה נפנה לחלקה הראשון של התוכנית, העוסק בקריאת המשתנים. לצורך כך ניצור תת-שיגרה שתבצע תהליך זה:

```
procedure Klita;
begin
   Writeln ('Enter three integer values (A, B, C):');
   Readln (A, B, C);
end;
```

תפקידו של חלק זה הוא אך ורק לקרוא את הנתונים. אם בעתיד נרצה לשנותו, לדוגמה, כך שיבחר או ייחשב ערכים במקום לקרוא אותם מהמשתמש, נוכל לעשות זאת ללא כל בעיה, כי התוכנית בנויה בצורה מודולרית. השינוי שנערוך בחלק זה של התוכנית לא צריך להשפיע על חלקים אחרים. כאשר כל חלק בתוכנית מבצע את תפקידו ללא תלות באופן פעולה של חלק אחר, ניתן "לשחק" עם כל אחד מהם כרצוננו.

כעת נפנה לחלק השני של התוכנית: מציאת הערך הגדול ביותר מבין השלושה:

```
procedure FindBig;
begin
   Largest := C;

   if  (A > B)  and  (A > C)  then
      Largest := A;

   if  (B > A)  and  (B > C)  then
      Largest := B;
end;
```

בתת-שיגרה זו אנו בודקים מיהו הערך הגדול ביותר. בתחילה אנו מניחים כי הערך של משתנה C הוא הגדול ביותר. במשפט התנאי הראשון אנו שואלים שאלה: אם תוכן המשתנה A גדול מתוכן המשתנה B וגם תוכן המשתנה A גדול מתוכן המשתנה C, אזי ניתן להסיק כי תוכן המשתנה A הוא הגדול ביותר. במשפט התנאי השני אנו שואלים שאלה דומה: אם תוכן המשתנה B גדול מתוכן המשתנה A וגם תוכן המשתנה B גדול מתוכן המשתנה C, אזי המשתנה B מכיל את הערך הגדול ביותר. בסיומה של תת-שיגרה זו נקבל במשתנה Largest את הערך הגדול מבין שלושת הערכים שנקלטו.

כעת, כל שנשאר לנו הוא להדפיס את תוצאות התוכנית. על כך אחראית תת-השיגרה בשם Print:

```
procedure Print;
begin
   Writeln ('Largest value is: ', Largest);
end;
```

קטעי התוכנית מודולריים, וכל קטע מבצע את המוטל עליו ללא התחשבות בפעולות שמתבצעות על ידי קטעי תוכנית אחרים. לכן ניתן יהיה גם כאן לשנות כל קטע מבלי לפגוע באחרים. לדוגמה, נוכל להוסיף צבעים לטקסט המוצג, לשנות את נוסח וצורת ההודעה ועוד.

עתה, לאחר שיצרנו את כל חלקי התוכנית, נכתוב אותה בשלמותה:

```
program Example;
var
   A, B, C:       Integer;
   Largest:      Integer;

procedure Klita;
begin
   Writeln ('Enter three integer values (A, B, C):');
   Readln (A, B, C);
end;

procedure FindBig;
begin
   Largest := C;
   if  (A > B)  and  (A > C)  then
      Largest := A;

   if  (B > A)  and  (B > C)  then
      Largest := B;
end;

procedure Print;
begin
   Writeln ('Largest value is: ', Largest);
end;

begin          {main program}
   Klita;
   FindBig;
   Print;
end.
```

נוכל לראות עד כמה **השיגרה הראשית** פשוטה. כל אשר עליה לעשות הוא לקרוא ברצף לתת-שגרות המבצעות את המטלה, זו אחר זו, ולקבל תוכנית שכתובה בצורה מודולרית.

למעשה, כל מה שלמדנו עד עכשיו הוא **מידור תוכנית - כתיבה מודולרית**. כלי זה הוא אחד מאבני היסוד לכתיבה נכונה של תוכניות בכל שפה. כל אשר נלמד עד כה בפרק זה משמש גם לצורך תכנות וגם להבנת נושאים נוספים. מכיון ששפת פסקל היא הנושא העיקרי, נפנה כעת לגרסה נוספת של תת-שיגרה.

יצירת פונקציות (Function declaration)

על הפונקציות למדנו בפרק 2. כבר אז מצאנו כי פונקציה היא גירסה של פרוצדורה, כלומר תת-שיגרה במבנה זהה עם שוני קטן: הפונקציה היא תהליך שבסופו אנו מקבלים ערך כלשהו - **הערך המוחזר** מהפונקציה.

כל מה שנאמר עד כה על מבנה ופעולה של פרוצדורה תקף גם לגבי פונקציה. לכן, כל שנשאר לנו לעשות הוא ללמוד כיצד מנצלים את התכונה החשובה שלה - **החזרת ערך**.

את הפונקציה מגדירים כך:

```
function FuncName (VarName : VarType1) : VarType2;
```

המילה השמורה function מציינת כי שורה זו כוללת הגדרת פונקציה. המילה **FuncName** מייצגת את שם הפונקציה. **VarName**, כמו בתת-שיגרה מסוג פרוצדורה, הוא שם המשתנה שדרכו מעבירים לפונקציה נתונים. VarType1 הוא סוג המשתנה הזה. ומה בדבר הערך המוחזר מהפונקציה? **סוג הערך המוחזר** נקבע על ידי VarType2. לדוגמה:

```
function Result (A : Integer) : Integer;
```

זוהי פונקציה בשם Result, הדורשת ערך שלם A ומחזירה ערך שלם. במהלך התוכנית "סוג הערך המוחזר" מהפונקציה הוא גם "סוג הפונקציה".

ניתן לקרוא לפונקציה מסוג זה בצורה הבאה:

```
program Example;
var
   X:          Integer;
   ...
begin
   ...
   X := Result (15);
   ...
end.
```

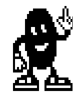
סוג הפונקציה יכול להיות כמעט מכל סוג נתון שהוא. בהמשך, כאשר נלמד סוגים נוספים של משתנים, נוכל לפרט אילו מהם אינם יכולים להיות ערך מוחזר מפונקציה. כל סוגי המשתנים שנלמדו עד כה מתאימים להגדרת פונקציה.

נתבונן כעת בתהליך החזרת ערך מפונקציה. הנה פונקציה לדוגמה:

```
function Big (A, B : Integer) : Integer;
begin
   if  A > B  then
      Temp := A
   else
      Temp := B;
   Big := Temp;
end;
```

Temp הוא משתנה שהוגדר כשלם בחלק ההצהרות של התוכנית.

לפונקציה זו שני ערכים מקומיים: A ו-B, שניהם מסוג שלמים וגם הפונקציה מוגדרת כשלם. חלקה הראשון של הפונקציה כולל תנאי המכתיב למחשב להציב ב-Temp את הערך הגדול מבין השניים. חלקה השני של הפונקציה הוא החשוב כאן. הוא זה שמורה למחשב כי על הפונקציה להחזיר את הערך שבמשתנה Temp. אנו רואים כי כדי להחזיר ערך כלשהו, אנו **מציבים אותו בשמה של הפונקציה,** כאילו היא המשתנה. כך למעשה מורים לפונקציה להחזיר ערך מסוים.



הערה: *לא ניתן לקרוא לפונקציה בדרך שקוראים לפרוצדורה. לדוגמה, את הפונקציה שזה עתה הצגנו, לא ניתן לקרוא כך:*

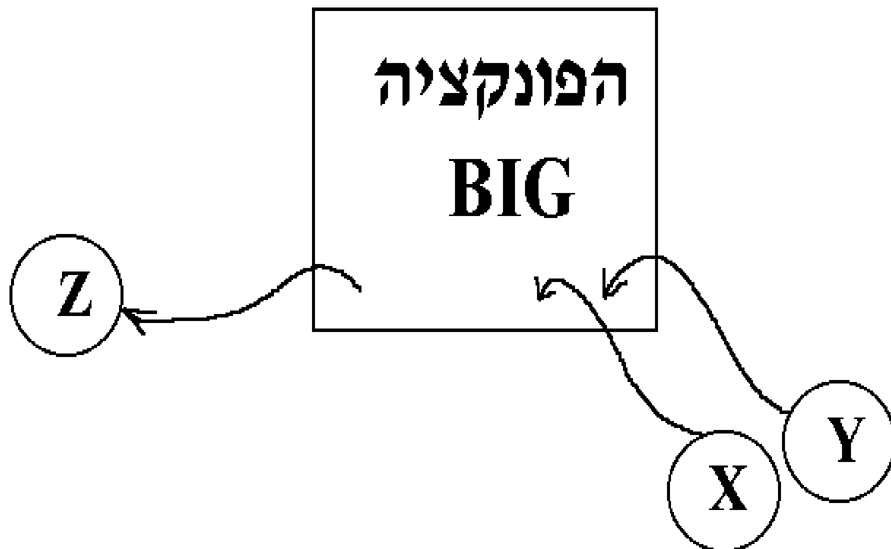
```
Big (X, Y);
```

צורת קריאה נכונה תהיה:

```
Z := Big (X, Y);
```

בדוגמה זו, הערך שנקבל מהפונקציה Big מועבר ל-Z. פירוש הדבר שכאשר אנו כותבים פונקציה אנו מנהלים ערך שמוחזר ממנה **והוא** משמש אותנו כמרכיב בביטוי.

נוכל לתאר את הפונקציה גם בתרשים:



בחרנו באופן שרירותי את שמות המשתנים X ו-Y שתפקידם לשמש כשני משתנים מקומיים הקולטים לתוכם שני ערכים שלמים מהתוכנית הראשית. ניתן לדמות שני משתנים אלה למשתנים A ו-B שבדוגמה האחרונה. בעזרת המשתנים הללו, שדרכם עוברים הערכים לפונקציה BIG, יודע המחשב מה להציג במוצא (פלט) של הפונקציה. ניתן לראות כי הנתון "זורם" החוצה לתוך המשתנה Z. משתנה זה אינו קשור לפונקציה, אלא מוגדר בדרך כלל כמשתנה גלובלי. את התרשים הזה ניתן לרשום כך:

```
Z := Big (1,3);
```

X ו-Y הם 1 ו-3 בהתאמה, ואילו ערך המוצא יתקבל ב-Z.

ננסה להסביר זאת בדרך אחרת. נתבונן בקטע התוכנית הבא:

```
....
function Big (X, Y : Integer) : Integer;
begin
   ....
end;

begin                { main program }
   Z := Big (A, B);
   Z := Big (Num1, Num2);
   Z := Big (Val1, Val2);
end;
```

המשתנים שברשימה זו מוגדרים כשלמים:

A, B, Z, Num1, Num2, Val1, Val2



שאלה: בדוגמה זו הגדרנו שני משתנים מקומיים בשמות X ו-Y השייכים לפונקציה Big. עתה, בכל שורה משלוש השורות של התוכנית קראנו לפונקציה זו עם זוגות שונים של משתנים. כיצד, אם כן, מתבצע הדבר?

תשובה: המשתנים X ו-Y משמשים לצורך ייצוג, כלומר, כאשר נציין למחשב כי אנו רוצים להשתמש במשתנים A ו-B לצורך הפעלת הפונקציה, המחשב יעתיק את הערך שבמשתנה A לתוך המשתנה X ובאותו אופן יעתיק את הערך שבמשתנה B לתוך המשתנה Y. כך גם יעשה לגבי זוגות המשתנים Num1, Num2 ו-Val1, Val2. בצורה זו אין הפונקציה "יודעת" באילו משתנים היא למעשה משתמשת, כי כל אשר מוטל עליה הוא לבצע חישוב או תהליך מסוים המתבסס על הנתונים שצוינו לה. מקובל לכנות בשם **משתנים אקטואליים** את המשתנים המופיעים כפרמטרים בפונקציות או בפרוצדורות.

ניהול משתנים חיצוניים

נניח כי הוטלה עלינו מטלה לכתוב תת-שיגרה שתפקידה לקלוט שני ציונים **ולאחסן אותם במשתנים כללים (גלובליים)**, שהמתכנת צריך להגדיר. במילים אחרות, עלינו לכתוב תת-שיגרה **שמציינים לה** שני משתנים והיא קולטת לתוכם ערכים. יסודות התכנות שנלמדו עד כה אינם מאפשרים לעשות זאת.
נניח שאנו מגדירים תת-שיגרה באופן הבא:

```
procedure GetMarks (A, B : Real);
```

ידוע לנו כי המשתנים A ו-B הם משתנים מקומיים ואין באפשרותם להשפיע על משתנים אחרים שמחוץ לתת-השיגרה הזו. אם כן, איך נוכל לבצע מטלה זו? גם על בעיה זו חשבו מתכנני שפת פסקל. לצורך פתרון הבעיה הוחלט כי כאשר תוסף המילה השמורה var, יוכלו משתנים מקומיים לשנות משתנים אחרים. לדוגמה:

```
procedure GetMarks (var A, B : Real);
begin
   Writeln ('Enter two real values: ');
   Readln (A, B);
end;
```

כאשר הוספנו את המילה השמורה var, המחשב "הבין" כי ברצוננו לשנות את המשתנים מחוץ לתת-השיגרה. כעת, כאשר נרצה לקרוא לתת-שיגרה זו עלינו לספק לה שני משתנים. אם ננסה להציב במקומם ערכים ממשיים, המחשב יודיע על תקלה.

נקרא לתת-שיגרה באופן הבא:

```
program Example;
var
   X, Y:          Real;

procedure GetMarks (var A, B : Real);
begin
   Writeln ('Enter two real values: ');
   Readln (A, B);
end;

begin
   GetMarks (X, Y);
end.
```

במקרה זה המשתנה A ייצג את המשתנה X, והמשתנה B ייצג את המשתנה Y. בצורה זו ניתן יהיה לשלוט על משתנים על פי בחירת המתכנת.

נעיין בדוגמה נוספת:

```
program FindBig;
var
   X, Y:          Integer;
   Temp:          Integer;

procedure Big (var A, B : Integer);
begin
   if  B > A  then
       begin
           Temp := A;
           A := B;
           B := A;
       end;
end;

```

```
16) begin
17)     Writeln ('Please enter two integer values:');
18)     Readln (X, Y);
19)     Big (X, Y);
20)     Writeln ('Higher value: ', X);
21)     Writeln ('Lower val: ', Y);
22) end.
```

בתוכנית זו הגדרנו שני משתנים (X ו-Y בשורה 3). בתחילת התוכנית (שורות 17 ו-18) אנו מציגים הודעה המבקשת מהמשתמש להכניס שני ערכים, אשר ייקלטו לתוך המשתנים X ו-Y. בשורה 19 אנו קוראים לתת-שיגרה Big תוך ציון שני המשתנים X ו-Y. מכיון שבהגדרת תת-השיגרה כללנו את המילה השמורה var, כל פעולה שתשנה את תוכן המשתנים המקומיים (A ו-B) תשנה גם את תוכן המשתנים שלנו (X ו-Y). בעזרת כלי חשוב זה נוכל לכתוב תת שגרות אשר מלבד זה שהן מקבלות ערכים לפעולה, הן יכולות להשפיע על משתנים שאנו, כמתכנתים, נוכל לקבוע, מבלי שנצטרך "לנבור" בשגרות הללו ולהכיר את שמות המשתנים שהוגדרו בהן.

שימוש חוזר בתת-שגרות

בתחילת הפרק, כאשר פרטנו את היתרונות של השימוש בשגרות אמרנו גם כי הן נועדו לקצר תהליכים. בדוגמה שלפנינו אנו רואים כי השימוש במילה השמורה var כדי "לשתף" משתנים מסוימים מחוץ לתת-שיגרה חוסך עבודה רבה. לדוגמה, במהלך התוכנית נוכל לקרוא פעמים נוספות לפרוצדורה Big, וכל פעם עם משתנים אחרים. בצורה זו לא נאלץ לכתוב כמה פרוצדורות, אחת לכל מקרה שניתקל בו, מכיון שלתת-שיגרה זו יש יכולת להשפיע חיצונית על משתנים שאנו נבחר.

קינון תת-שגרות (Sub-nesting)

למעשה, קינון תת-שגרות אינו נושא חדש. הרעיון בקינון הוא, שניתן לקרוא לתת-שיגרה אחת מתוך תת-שיגרה אחרת. כיצד, אם כן, נעשה הדבר? התשובה לכך פשוטה, קריאה לתת-שיגרה אחת מתוך תת-שיגרה אחרת נעשית בדרך שלמדנו: הצגת השם והנתונים (אם נדרש) שעל תת-השיגרה לקבל.

לדוגמה:

```
program Example;

procedure WriteValue (I : Integer);
begin
   Writeln ('Value is: ', I);
end;

procedure WriteList;
var
   I:          Integer;

begin
   for I:=1 to 10 do
         WriteValue (I);
end;

begin
   WriteList;
end.
```

בתוכנית זו הגדרנו שתי תת-שגרות. הראשונה שמה WriteValue ותפקידה להדפיס את הנתון השלם שמועבר לה. תת-השיגרה השנייה היא WriteList והיא כוללת בתוכה לולאה שבעזרתה השיגרה WriteValue נקראת עשר פעמים. יש לשים לב, כי שני המשתנים שמוגדרים בשתי תת-השגרות הינם מקומיים, ורק תת-השיגרה ששייכת אליהם יכולה לטפל בהם. כאשר הלולאה בתת-שיגרה WriteList קוראת לתת-שיגרה WriteValue היא מעבירה לה ערך מסוים. מבחינתה של השיגרה WriteValue, הערך שמתקבל במשתנה I הוא ערך שלם פשוט, ואין זה משנה לה מהיכן הגיע. בצורה זו ניתן לפשט ברמה נוספת את התוכנית. במידה ויש תת-שגרות שמבצעות את אותו התהליך, ניתן ליצור תת-שיגרה שכאשר יקראו לה, היא תבצע אותו.

נוהל זה פשוט למדי, **אך** יש בו מלכוד קל: לדוגמה, נחליף את סדר ההופעה של תת-השגרות:

```
procedure WriteList;
var
   I:          Integer;

begin
   for I:=1 to 10 do
      WriteValue (I);
end;

procedure WriteValue (I : Integer);
begin
   Writeln ('Value is: ', I);
end;
```

במקרה זה המחשב יודיע על שגיאה. מקור הבעיה הוא, שניתן לקרוא אך ורק לתת-שגרות שהוגדרו קודם לתת-שיגרה שקוראת להם. מסיבה זו, גם ניתן לקרוא בשיגרה הראשית לכל תת-השגרות, מכיון שאלו כבר הוגדרו בתחילת התוכנית, לפניה.

תרגילים

תרגיל 6.1

כתבו תוכנית הקולטת עשרה ציוני תלמידים ומחשבת את ממוצע הציונים האלה. על התוכנית להכיל לולאה בשיגרה המרכזית של התוכנית ותת-שיגרה הקולטת את הציונים שעורכת ביניהם את הממוצע.

תרגיל 6.2

כתבו תוכנית הקולטת שני ערכים שלמים. על התוכנית להדפיס את הגדול מבין שניהם בעזרת תת-שיגרה שתבחן אותם.

תרגיל 6.3

כתבו תוכנית המגדירה משתנה שלם מתוחל מראש לערך 0 (יש להשתמש בהגדרת const). על התוכנית לנוע בלולאה הקוראת ערך מהמשתמש. המשך הלולאה יכלול קריאה לתת-שיגרה שתשווה את הערך שזה עתה נקרא לערך הקודם שנקרא. אם הערך החדש קטן מהערך הקודם לו, תת-השיגרה תדפיס הודעת שגיאה. בכל מקרה אחר תת-השיגרה תעדכן את המשתנה המתוחל מראש לערך החדש. הלולאה בשיגרה הראשית תימשך עד אשר תקלוט ערך שלילי.

תרגיל 6.4

בהסתמך על הפונקציה **big** שהצגנו בפרק זה, כתבו תוכנית הקולטת שלושה ערכים ומדפיסה את הגדול מביניהם. נסו לערוך השוואה בין יעילות התוכנית הזו, אל מול הדוגמה הדומה לה בתחילת הפרק.

תרגיל 6.5

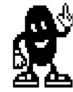
כתבו תוכנית המחשבת ממוצע ציונים של כיתה שלמה. התוכנית תכלול שלוש תת-שגרות: הראשונה היא תת-שיגרה מסוג פונקציה, שתפקידה לקלוט ולהחזיר את מספר הכיתות שלהן אנו רוצים לחשב את הציונים. תת-שיגרה שנייה, גם היא מסוג פונקציה, שתפקידה יהיה לקלוט כל פעם כמה תלמידים יש בכיתה, ותת-שיגרה שלישית המכילה לולאה כדי לקרוא את הציונים של כיתה מסוימת ולהדפיס את הממוצע.

תרגיל 6.6

כתבו תוכנית הכוללת פונקציה שתפקידה לקבל את שלושת הקבועים של משוואה ריבועית (ax^2+bx+c) ולחשב את הדיסקרימיננטה ($b^2 - 4ac$). במקרה שלא ניתן לחשב ביטוי זה, על הפונקציה להחזיר את הערך 1-. התוכנית תקלוט את שלושת הערכים הללו, תקרא לפונקציה ותדפיס את התוצאה.

תרגיל 6.7

כתבו תוכנית הקולטת ערך שלם ומדפיסה את סכום ספרותיו. התוכנית תכלול תת-שיגרה שתפקידה לקבל ערך זה ולחשב את הסכום.



הערה *: הערך יכול להיות בן מספר מסוים של ספרות, לכן יש ליצור לולאה שתחשב את הסכום.*

(**רמז:** יש להשתמש באופרטורים mod ו-div שלמדנו בפרק 2).

תרגיל 6.8

כתבו תוכנית הכוללת פונקציה שתפקידה לחשב את הערך המקורב של הקבוע Pi (...3.1415). ערך זה מחושב על-ידי הטור:

$$Pi = 4 * (\frac{1}{1} - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - ... \pm \frac{1}{(2N-1)} ....)$$

הפונקציה תסיים את החישוב כאשר האיבר האחרון המחושב קטן מ-0.0001.

תרגיל 6.9

כתבו תוכנית הקולטת משכורת של עובד ואת אחוז המס שיש לנכות ממנה. התוכנית תפעיל תת-שיגרה שתחשב את המשכורת נטו (לאחר ניכוי המס). את התוצאה יש להציב במשתנה שיצוין על ידי המתכנת.

(**רמז:** יש להשתמש במילה השמורה var).

פרק 7

מחרוזות

מהי מחרוזת (string)?

בחיי היומיום אין אנשים מדברים בינם לבין עצמם במספרים. התקשורת וההבנה ביניהם נעשית על ידי שמות ומשפטים. גם מחשב PC שאנו משתמשים בו מבין אך ורק מספרים. כדי ליצור מחשב פשוט, חייבת שפתו להיות פשוטה ולכן, המחשב "מדבר" אך ורק בדרך זו. מאידך, בני האדם אינם יכולים לתקשר ביניהם, או בינם לבין המחשב, במספרים בלבד.

לשם כך פותחו שפות תכנות, ביניהן **טורבו פסקל (Turbo Pascal)**, שתפקידן לגשר בין המשתמש לבין המחשב. אחד מאמצעי הגישור בין המשתמש לבין המחשב הוא **המחרוזת** (**string**).

מכיון שלא ייתכן כי נתונים כמו שם פרטי, שם משפחה, רחוב, הערות וכדומה, יועברו למחשב בצורת מספרים, שפת פסקל מספקת לנו כלים לעשות זאת: זוהי **המחרוזת**, שתפקידה להכיל רצף של תווים.

מבנה מחרוזת בשפת פסקל

מסיבות הסטוריות נקבע כי כל מחרוזת תוכל להכיל רצף של עד 255 תווים, כלומר: שם, הערות, או אפילו סתם נתונים אקראיים שאורכם לא עולה על 255 סימנים. המבנה הבסיסי שעל פיו מגדירים **משתנה מסוג מחרוזת** הוא:

```
var
   var_name:      string;
```

"**var_name**" הוא שם המשתנה, ואילו "**string**" זוהי המילה להגדרת מחרוזת. לדוגמה:

```
var
   Name:          string;
```

הגדרה זו מאפיינת משתנה בשם "**Name**" כמשתנה מחרוזת, שאורכה המירבי יכול להגיע ל-255 תווים. בדרך כלל, אנו לא זקוקים לאורך המירבי האפשרי, ועל כן נרצה להגדיר את האורך המירבי לכל משתנה בנפרד. כך נוכל גם לחסוך בזיכרון. נוכל להגדיר אורך רצוי של מחרוזת על ידי התבנית הבאה:

```
var
   var_name:      string[length];
```

התוספת בתבנית זו היא המילה **length**, אשר מגדירה את האורך המירבי של המחרוזת. לדוגמה,

```
var
   Str1:          string[15];
```

זוהי הגדרה של משתנה מחרוזת שיוכל להכיל בתוכו עד 15 תווים.

הערה: *אורך מחרוזת יכול להיות אך ורק בתחום של 0 עד 255.*

פעולות מחרוזת בסיסיות

הטיפול במחרוזות

שפת פסקל מאפשרת שימוש קל ונוח במחרוזות. ניתן לקלוט שם או פרטים של אדם או עצם לתוך מחרוזת באותה פשטות שבה השפה קולטת נתונים אחרים כמו שלמים, מספרים ממשיים, תווים ועוד. הדוגמה הבאה מדגימה עיקרון זה:

```
program Example1;
var
   Age:          Integer;
   Name:         string;

begin
   Writeln ('מהו שמך?');
   Readln (Name);
   Writeln ('מהו גילך?');
   Readln (Age);
end.
```

תוכנית זו קוראת שם לתוך משתנה מחרוזת ולאחר מכן היא קולטת גיל לתוך משתנה מסוג שלם. אם היינו משנים את הגדרת משתנה המחרוזת ל-**Name string[6]**, השם שהיינו מכניסים היה מוגבל ל-6 אותיות בלבד. כמובן שהשפה לא היתה מודיעה על שגיאה אם היינו מכניסים שם ארוך יותר מ-6 אותיות, אך המשתנה היה קולט את 6 האותיות הראשונות בלבד, ומתעלם מהשאר.

בנוסף לדרך שבה פסקל קולטת מחרוזות, ניתן להציב ב"משתנה מחרוזת" ערך קבוע מראש. דבר זה נעשה כך:

```
var_name  :=  ' מחרוזת '
```

כל מחרוזת קבועה, חייבת להתחיל ולהסתיים בגרש בודד ('), כשם שראינו במשפט של הפקודה Writeln. קטע התוכנית הבא מדגים זאת:

```
program Example2;
var
   St:          string;

begin
   St:= 'שלום !!!';
   Writeln (St);
end.
```

השוואת מחרוזות

משתנה המחרוזת הוא משתנה ככל משתנה אחר. בשל כך ניתן להשוות את תוכן המשתנה לערך כלשהו או למשתנה מחרוזת יותר מאשר כשאנו משווים שני משתנים שלמים. ההשוואה נעשית באופן הבא:

```
if  string1 = string2  then ...
```

טיפול בתווים בודדים

במקרים מסוימים יש צורך לשנות או לקרוא תו בודד מתוך מחרוזת. כיצד נעשה הדבר?



שימו לב: *הביטוי* ***string[position]*** *מאפשר להתייחס לתווים בודדים בתוך מחרוזת שלמה, על פי המיקום שלהם בתוך המחרוזת. ספירת התווים במחרוזת הינה מצד שמאל!*

לדוגמה, הביטוי 'St[1]:='A ישנה את התו הראשון במחרוזת St. באותו אופן, אם נכתוב (Writeln (St[3] ידפיס המחשב רק את התו השלישי מתוך המחרוזת St. למעשה, כל דבר שניתן ליישם על משתנה מסוג **תו** (**CHAR**), ניתן ליישם על תווים בודדים בתוך מחרוזת. תוכנית דוגמה:

```
program Example3;
var
   St:          string;

begin
   Writeln ('הקלד מחרוזת כלשהי:');
   Readln (St);
   Writeln ('St[1] = ', St[1]);
end.
```

פונקציות מחרוזת נוספות

לא נוכל לספק את כל דרישות התכנות של מחרוזות באמצעות הפעולות הבסיסיות שלמדנו עד כה. פעולות כמו חיפוש מילה בתוך מחרוזת, או עריכת מחרוזת הן פעולות שנעשה בהן שימוש רב. כל מה שלמדנו עד כה לא יכול לספק לנו את הכלים המתאימים לכך. שפת פסקל כוללת כלים מתקדמים יותר לטיפול בנושאים אלה.

הפונקציה Length (אורך)

אחת הפונקציות השימושיות ביותר היא **Length**. הפונקציה מקבלת מחרוזת כנתון ומחזירה את אורכה. תוכנית דוגמה:

```
program Example4;
var
   I:          Integer;
   St:         string;

begin
   Writeln ('Type in a string:');
   Readln (St);
   I := Length(St);
   Writeln ('String length = ', I);
end.
```

תוכנית זו קולטת מחרוזת ומדפיסה את אורכה.

הפונקציה Copy (העתקת קטע מחרוזת)

במקרים מסוימים יש צורך להעתיק חלק ממחרוזת. לשם כך מספקת שפת פסקל את הפונקציה **Copy**, שתפקידה להעתיק חלק ממחרוזת קיימת. השימוש בפונקציה נעשה בתבנית הבאה:

```
string := Copy (Str, Index, Count)
```

Str - המחרוזת שממנה רוצים להעתיק קטע.

Index - המספר הסידורי של התו שהחל ממנו תתחיל ההעתקה.

Count - מספר התווים שרוצים להעתיק.

לדוגמה:

```
St := Copy ('GOOD BYE', 1 ,4)
```

במשפט זה מעתיקים את המילה "GOOD" לתוך המשתנה St.

הפונקציה Pos (מיקום תת-מחרוזת)

במקרים מסוימים עלינו לחפש מילה או קבוצת תווים מסוימת בתוך מחרוזת. לצורך כך קיימת פונקציה מיוחדת שתפקידה הוא "לומר" לנו היכן ממוקמת תת-מחרוזת בתוך מחרוזת נתונה. הפונקציה היא **Pos**. אופן השימוש בה:

```
int_var := Pos (Sub_Str, Str)
```

Sub-Str - תת-המחרוזת שמחפשים בתוך המחרוזת.

str - המחרוזת שבה מחפשים.

int_var - משתנה זה יקבל את מיקום תת-המחרוזת בתוך המחרוזת.

לדוגמה, נכתוב את המשפט הבא:

```
I = POS ('TO', 'HELLO TO YOU') ;
```

הפקודה תציב במשתנה I את הערך 7, מכיון שהמחרוזת "TO" מצויה בתוך המחרוזת "HELLO TO YOU" החל מהתו השביעי.

נבדוק לדוגמה, מצב אחר:

```
I := POS ('X', 'ABCDEF');
```

במקרה זה תת-המחרוזת אינה קיימת בתוך המחרוזת ולכן I יקבל את הערך 0.

שימו לב: *הפונקציה POS מחזירה **תמיד** את מיקום הצירוף **הראשון** שהיא מוצאת, ואינה ממשיכה לחפש במחרוזת צירופים נוספים.*

שינוי מבנה מחרוזת

הפקודה Insert (הכנסת תת-מחרוזת)

למדנו שניתן לחבר שתי מחרוזות על ידי תבנית המשפט הבאה:

```
St1 :=  St1  +  St2
```

אך מה קורה כאשר רוצים "לשתול" מחרוזת אחת בתוך מחרוזת אחרת? לשם כך קיימת פקודה שימושית, שבעזרתה ניתן לעשות זאת. מבנה הפקודה:

```
Insert (SubStr, Str, Index);
```

SubStr - המחרוזת שאנו רוצים לשתול.

Str - המחרוזת שבה אנו שותלים את המחרוזת המשנית.

Index - המיקום שממנו אנו רוצים להתחיל לשתול את המחרוזת המשנית בתוך המחרוזת הראשית.

לדוגמה:

```
program Example5;
var
   Name, St:      string;

begin
   St := 'Hello,  , how are you today?';
   Writeln ('Enter your name:');
   Readln (Name);
   Insert (Name, St, 7);
   Writeln (St);
end.
```

תוכנית זו מקבלת כקלט את שם המשתמש "ושותלת" אותו במיקום ה-7 במחרוזת, אחרי הפסיק העוקב את המילה "HELLO,". אם נכניס, לדוגמה, את השם "AVI", המחשב ידפיס את ההודעה:

```
Hello,AVI, how are you today?
```

הפקודה Delete (מחיקת קטע ממחרוזת)

הפקודה "הנגדית" ל-**INSERT** היא **DELETE**. תפקידה למחוק קטע מתוך מחרוזת. מבנה הפקודה:

Delete (Str, Index, Count);

Str - המשתנה שמכיל את המחרוזת.

Index - המיקום במחרוזת שממנו מתחילים במחיקה.

Count - מספר התווים שיש למחוק מתוך המחרוזת.

לדוגמה, אם נתונה המחרוזת "WHXXAT IS YOUR NAME?", וברצוננו למחוק את שתי אותיות X שבמילה "WHXXAT", נרשום:

Delete (St, 3, 2);

(St זה הוא משתנה המכיל את המחרוזת, 3 - המיקום לתחילת המחיקה, ו - 2 מספר התווים שצריך למחוק). דוגמה:

```
1)   program Example6;
2)   var
3)        St:       string;
4)        Index:    Integer;
5)
6)   begin
7)        Writeln ('הכנס מחרוזת כלשהי, שבה לפחות שתי מילים:');
8)        Readln (St);
9)        Index := Pos ('  ',  St);
10)       if  Index <> 0  then
11)           begin
12)                  Delete (St, 1, Index);
13)                  Writeln ('מחרוזת ללא המילה הראשונה: ');
14)                  Writeln (St);
15)           end
16)       else
17)           Writeln ('שגיאה: המחרוזת חייבת להכיל לפחות שתי מילים.');
18)  end.
```

תוכנית דוגמה זו קצת יותר מורכבת, אך כאשר ננתח אותה, נראה שהיא פשוטה למדי. בשורות 3 ו-4 מוגדרים שני משתנים: St מסוג מחרוזת, ו-Index מסוג שלם.

תפקיד שורה 9 למצוא את מיקום הרווח הראשון, כלומר סוף המילה הראשונה. תפקיד משפט התנאי (... if... then) בשורה 10 לבחון אם קיימת יותר ממילה אחת, שהרי אם היתה רק מילה אחת, לא היה רווח (בין המילה הראשונה

לשנייה). כשהפונקציה POS אינה "מוצאת" את המחרוזת המבוקשת (במקרה שלנו תו רווח), היא מחזירה ערך "0". תפקיד Delete בשורה 12 הוא למחוק מן התו הראשון (1) ועד למיקום הרווח (כולל הרווח).

תפקיד שורות 13 ו-14 להדפיס הודעה, ולאחר מכן להדפיס את המחרוזת לאחר עריכתה. אם התנאי לא מתקיים (לא נמצא רווח, והפונקציה POS החזירה ערך "0") המחשב ידפיס הודעת שגיאה.

תוכנית דוגמה נוספת:

תוכנית זו יותר מורכבת ואף נראית מסובכת, אך ניתוח מפורט של שורות הקוד מבהיר את משמעותה ואופן פעולתה. תפקיד התוכנית לקרוא מחרוזת ולהפוך את סדר התווים, כדי שתודפס מן התו האחרון לתו הראשון. למשל, המחרוזת המקורית היא ABC וההדפסה תהיה CBA.

```
1)    program Example7;
2)    var
3)        St1, St2:    string;
4)
5)    begin
6)        Writeln ('הכנס מחרוזת:');
7)        Readln (St1);
8)
9)        St2 := ' ';
10)       while  Length (St1) > 0  do
11)           begin
12)               Insert (St1[1], St2, 1);
13)               Delete (St1, 1, 1);
14)           end;
15)       Writeln ('המחרוזת לאחר עריכתה:');
16)       Writeln (St2);
17)   end.
```

בשורה 3 מוגדרים שני משתנים: St1 המשמש כמחרוזת המקור, המחרוזת הראשונית. תפקיד המשתנה St2 הוא לאחסן בגמר התהליך את המחרוזת ההפוכה. הרעיון שעליו מבוססת התוכנית הוא שימוש בפקודה Insert כדי להכניס תו חדש לתוך מחרוזת היעד St2 ולאחר מכן למחוק אותו מ-St1. לכן, תפקיד שורה 10 לבדוק אם אורך המחרוזת St1 (שאותה מקצצים) שווה ל-0. לדוגמה, אם ב- St1 היתה המילה "HELLO", התוצאה המתקבלת ב-St2 ובהדפסה היתה: "OLLEH".

מחרוזות ומספרים

תוכנה מורכבת היא לא רק תוכנה אשר קיימים בה משתנים רבים וחישובים רבים. תוכנה מורכבת היא תוכנה שקיימים בה קשרים רבים בין מחרוזות, משתנים, משפטי תנאי, לולאות ועוד. במקרים רבים מתקבל נתון מספרי כלשהו, אשר שמור במבנה של מחרוזת, ויש צורך להמיר אותו לערך מספרי מסוים, או להיפך. נניח לדוגמה, שנתון המשפט "I AM 21 YEARS OLD", ואנו רוצים לחשב את הגיל בעוד מספר שנים. יש ברשותנו כלים למחוק את כל אשר אינו נחוץ לנו על פי הידע שרכשנו עד כה. אך מה בדבר הערך "21"? כיצד נוכל לבצע עליו פעולות אריתמטיות (חישובים), שהרי פעולות מסוג זה דורשות משתנה מסוג שלם או ממשי. ולהיפך, כיצד ניצור מחדש מחרוזת כזו? כיצד נתרגם ערך מספרי למבנה מחרוזת? לצרכים אלה מספקת פסקל שתי פקודות יעילות - **Str** ו-**Val** שתפקידן לבצע מטלות אלו.

הפקודה Str (תרגום ערך מספרי למחרוזת)

פקודה זו מקבלת ערך מספרי, באמצעות משתנה כמובן, וממירה אותו למבנה מחרוזת. מבנה הפקודה:

```
Str (Value, String);
```

Value - הערך השלם או הממשי שאנו רוצים להמיר.

String - משתנה המחרוזת שבתוכו תאוחסן המחרוזת החדשה שנוצרה.

תוכנית לדוגמה:

```
program Example8;
var
   Age:       Integer;
   St:        string;

begin
   Writeln ('מהו גילך?');
   Readln (Age);
   Age := Age  +  4;
   Str (Age, St);
   WriteLn (ST, ' בעוד 4 שנים הגיל יהיה');
end.
```

תוכנית זו מקבלת כנתון את הגיל, מחשבת את ערכו בעוד ארבע שנים, מתרגמת אותו למשתנה מחרוזת ומדפיסה אותו.

הפקודה Val (הפיכת מחרוזת לערך מספרי)

לעיתים יש צורך להמיר ערך השמור במבנה של מחרוזת, לערך של משתנה מספרי. לשם כך קיימת הפקודה **VAL**. תבנית הפקודה:

Val (Str, Var, Index);

Str - משתנה מחרוזת שמכיל את הערך המספרי.

Var - משתנה שיכול להיות שלם או ממשי, ומאחסן את הערך בסיום ההמרה.

Index - משתנה מסוג שלם שתפקידו לדווח אם המחרוזת מתאימה להמרה או לא.

לדוגמה, לא ניתן להמיר את המחרוזת "1A34" מכיון שהיא כוללת את התו "A" (שאינו מספרי). במקרה זה המשתנה Index יכיל את מיקום התו השגוי, שבדוגמה זו הוא 2. אם ההמרה מתבצעת ללא טעויות, המשתנה Index מכיל את הערך 0.

תרגילים

תרגיל 7.1

כתבו תוכנית הקולטת שם ומדפיסה הודעה בנוסח: "שיהיה לך יום טוב, [שם נתון] ".

תרגיל 7.2

כתוב תוכנית שמכילה את המילה "HELLO" במשתנה מחרוזת **St1**, וקולטת מילה לתוך משתנה מחרוזת **St2**. אם שתי המחרוזות שוות, התוכנית מדפיסה את ההודעה: "HELLO TO YOU TOO".

תרגיל 7.3

כתבו תוכנית שתפקידה לקלוט מחרוזת ומספר שלם, ולהודיע על איזה תו מצביע המספר השלם. לדוגמה, אם נתונה המחרוזת "ABCDE", אם נזין את המספר 3 הוא יגרום להצגת ההודעה הזו על המסך: "ST[3] = C".

תרגיל 7.4

כתבו תוכנית אשר קולטת מחרוזת, ומדפיסה אותה בריווח כפול על המסך. לדוגמה, המחרוזת "HELLO" תודפס בצורה הבאה: "H E L L O". כתבו תוכנית זו תוך שימוש בפרוצדורה.

תרגיל 7.5

כתבו תוכנית אשר דומה לתרגיל 7.3. התוכנית תקלוט מחרוזת ומספר שלם. אם המספר נמצא בתחום אורך המחרוזת, התוכנית תדפיס את התו שאליו מצביע השלם. אם השלם גדול מאורך המחרוזת, התוכנית תדפיס "INVALID VALUE".

תרגיל 7.6

כתבו תוכנית הקולטת מחרוזת ומדפיסה את שתי האותיות האחרונות שבה.

(**רמז:** יש לבדוק לפני כן אם אורכה של המחרוזת הוא לפחות 2 תווים.)

תרגיל 7.7

כתבו תוכנית הקולטת משפט ומדפיסה את המילה הראשונה בלבד. על התוכנית לכלול תת-שיגרה הקולטת את המחרוזת ותת-שיגרה המדפיסה את המילה.

(**רמז**: יש להשתמש בפונקציות POS ו-COPY, וגם לבדוק שהפונקציה POS אינה מחזירה ערך 0).

תרגיל 7.8

כתבו תוכנית הקולטת מחרוזת, שבה אמור להיות ערך מספרי. אם המחרוזת שהתקבלה היא אכן ערך מספרי טהור ללא תווים מיותרים, התוכנית תדפיס הודעה בנוסח "הכל בסדר". אם המחרוזת אינה "תקינה", התוכנית תדפיס הודעה מתאימה ואת התו השגוי במחרוזת.

תרגיל 7.9

כתבו תוכנית הקולטת מספר שלם ומדפיסה אותו עם הפרדה של פסיק לאחר כל שלוש ספרות. לדוגמה, נתון 14532, ולכן פלט המחשב יהיה 14,532. התוכנית תכלול פונקציה שתקבל כנתון את הערך הנקלט ותחזיר מספר ערוך עם הפרדה של פסיקים.

תרגיל 7.10

כתבו תוכנית הקולטת ערך שלם, ממירה אותו למחרוזת ומחשבת את סכום ספרותיו. החישוב ייעשה בעזרת פונקציה מתאימה.

כתבו תוכנית נוספת הקולטת ערך שלם, ממירה אותו למחרוזת ומדפיסה אותו מן הסוף להתחלה.

תרגיל 7.11

כתבו תוכנית הקולטת מחרוזת "ומסלקת" כל אות החוזרת על עצמה יותר מפעם אחת ברצף. תהליך זה ייעשה על ידי פונקציה. לדוגמה, מחרוזת כגון:

" WHAATT DOO YOU MEEAN???"

תודפס כך:

"WHAT DO YOU MEAN?"

תרגיל 7.12

כתבו תוכנית הקולטת מחרוזת ומדווחת:

א. אורך המחרוזת.

ב. כמה מילים יש במחרוזת.

ג. מהי המילה הארוכה ביותר.

ד. מהי המילה הקצרה ביותר.

ה. מהו אורך המילה הממוצעת.

תרגיל 7.13

כתבו תוכנית הקולטת מחרוזת מסוימת ומצמצמת בה את כל הרווחים הכפולים, מוחקת את הרווח הראשון (אם קיים) ואת הרווח האחרון (אם קיים).

לדוגמה, נתונה המחרוזת: " שלום   לכל המשתמשים   ". לאחר עיבוד המחרוזת נקבל: "שלום לכל המשתמשים".

פרק 8

מערכים

מערך (**array**) הוא רצף של נתונים (איברים), אשר כולם מאותו סוג נתון. לכל **איבר** יש מספר סידורי, אשר מייצג את מיקומו במערך, ולכן לכל איבר ניתן לגשת בצורה ישירה, ללא כל תלות באיבר או באיברים אחרים.

מערכים חד-מימדיים

ניתן להציג מערך מן הסוג הפשוט ביותר כטבלה. **טבלה** מסוג זה יכולה להכיל רשימה של מספרים שלמים, שברים ואפילו מחרוזות. בדוגמה הבאה קיים מערך ובו שישה ערכים שלמים:

המערך TAB והאיברים

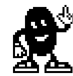
1	איבר מס' 1
13	איבר מס' 2
22	.
4	.
9	.
17	איבר מס' 6

לכל מערך (טבלה) יש **שם המייחד אותו.** שם המערך שבדוגמה שלנו הוא TAB, ולכל המשתנים בו יש אותו שם - TAB. המאפיין היחיד המבדיל איבר אחד ממשנהו הוא **המספר הסידורי.** לדוגמה, האיבר TAB[3] הוא השלישי במערך, ואם נבדוק את ערכו הנוכחי (הכוונה ל-TAB[3]) נקבל את הערך 22.

ניתן לדמות מערך למקרה בחיי היומיום. המחשב מטפל במערכים באותו אופן שהדוור מחלק מכתבים בשכונה שבה יש בתים פרטיים. הדוור בודק את שם הרחוב (המחשב קורא בהתאמה את שם המערך). לאחר מכן הוא קורא את מספר הבית (המחשב, לחילופין, בוחן את המספר הסידורי של האיבר). בצורה זו הדוור יודע את יעדו של המכתב וכך גם המחשב יודע את מיקומו של האיבר.

לשם מה להשתמש במערכים? במה זה מייעל את התוכנית? למעשה, יש תוכניות אשר בלעדי המערך לא היה ניתן לפתור אותן בכלל, ואם אפשר היה לעשות זאת, הפתרון היה מורכב ומסובך.

ניקח לדוגמה קלט ערכים מהמקלדת. אפשר לקלוט כל ערך למשתנה נפרד: A, B, C וכד', או שאפשר לקלוט את **כולם** לתוך **איברים של מערך אחד**. התנאי הוא כמובן, ש**כולם** יהיו מאותו סוג נתון: מספרים שלמים, מחרוזות וכד'. כאשר עוסקים במספר רב של נתונים, השימוש באיברים בודדים ושונים אינו מעשי כלל, ובשלב זה, טרם עסקנו בתכנות. נצטרך לכתוב תוכנית ענק שתטפל בכל איבר בנפרד.



הערה: *בפרק זה נעסוק אך ורק במערכים חד-מימדיים. בפרק הבא נסביר מהם מערכים רב-מימדיים, כיצד מגדירים ומשתמשים בהם.*

הנה דוגמה נוספת: לפנינו תוכנית שתפקידה לקלוט ציונים של מספר תלמידים. ללא הגדרת מערך ניאלץ לכתוב לולאה שתיראה כך:

```
for  I := 1  to  10 do
    begin
        Writeln ('Enter mark number ', I, ':');
        if  I = 1  then  Readln (Var1);
        if  I = 2  then  Readln (Var2);
                    :
        if  I = 10  then Readln (Var10);
    end;
```

Var1, Var2 וכו' הם הציונים של התלמידים השונים. בתוכנית זו נאלצנו לקרוא עשרה ערכים אל תוך עשרה משתנים שונים. ומה היה אילו צריך היה לקרוא 1000 ערכים?

איך משתמשים בתכונות המערך כדי לכתוב תוכנית יעילה וקצרה - זאת נלמד עכשיו.

הגדרת המערך בשפת פסקל

לפני שניגש לאופן הגדרת המערך בשפת פסקל, נבהיר תחילה כמה מן התכונות הבסיסיות הנחוצות להגדרה.

1. לכל מערך יש **שם** המייחד אותו, באותו אופן שלכל משתנה מסוג שלם, מחרוזת וכו', יש שם המייחד אותו משאר המשתנים.
2. **מספר האיברים קבוע** מראש. לא ניתן לשנות מספר זה בזמן ההרצה של התוכנית.
3. **סוג המשתנה** שעליו מבוסס המערך.

אופן הגדרת המערך:

```
var
   name:          array [m..n]  of  [type];
```

name - שם המשתנה המייצג את המערך.

array - מילה שמורה המגדירה לשפת פסקל כי המשתנה הוא מסוג מערך.

[m..n] - הגדרה זו קובעת את **תחום ההגדרה** של המערך (ולמעשה, גם את מספר האיברים). לדוגמה, הגדרת [2..10] תאפשר להשתמש במערך המתחיל מ-2 ומסתיים ב-10. גישה לאיבר מחוץ לתחום זה תגרום ל**שגיאה** בתוכנית.

of [type] - הגדרת **סוג** המשתנה הבסיסי שעליו מבוסס המערך.

לדוגמה:

```
var
   Marks:         array [1..10]  of  Integer;
```

זוהי הגדרת מערך ששמו Marks. תחומו הוא 1 עד 10 והוא מבוסס על משתנה מסוג שלם. על פי ההסבר בתחילת הפרק, המשתנה Marks הוא למעשה טבלה בעלת עשרה איברים מסוג שלם.

תחום המערך [m..n] יכול להיות בטווח של 65535- עד 65535. לא ניתן להגדיר ערך גבוה או נמוך משני ערכים אלה. את תחום המערך כותבים במבנה זה:

[ערך נמוך .. ערך גבוה]

במקרים מסוימים לא יהיה ניתן להגדיר מערכים גדולים יותר מדי, אפילו אם הם בתחום המותר, עקב מגבלות זיכרון.

דוגמאות לסוגים נוספים של מערכים:

```
var
   Tavim:        array [0..19]  of  Char;
   Values:       array [-3..15] of  Real;
   OK:           array [2..14]  of  Boolean;
   Names:        array [1..15]  of  string;
```

אפשר לראות את איברי המערך כאוסף של תאים בטבלה, וכך גם נתייחס אליהם בהמשך. נבחן שוב את צורת הטבלה, ונראה כי למעשה כל איבר "סגור" בתא משלו, מבודד מן האחרים. יש שוני בין תא פשוט לבין איבר, אך ניתן לומר כי בשימושים כשלנו, העוסקים במשתנים פשוטים, המושג תא הוא הנכון בשימוש. ההגדרה של איבר מיוחסת למשתנים הרבה יותר מורכבים שעדיין לא למדנו.

שימוש מעשי במערכים

עד כה למדנו מהם מערכים, למה הם משמשים וכיצד מגדירים אותם. למעשה, למדנו הכול חוץ מהדרך שבה מיישמים את השימוש בהם. על פי הגדרת המערך, כל איבר נבדל ממשנהו על ידי מספרו הסידורי. כדי לגשת לאיבר מסוים, דרוש שם המערך והמספר הסידורי של איבר שרוצים לפנות אליו.

כדי לגשת לאיבר בודד בתוך מערך מסוים נשתמש בתבנית הבאה:

```
var_name[position]
```

var_name - שם המערך

position - המספר הסידורי של האיבר.

סריקת מחרוזת

קליטת מספר רב של ערכים אל תוך מחרוזת בצורה פרטנית כמו זו שמוצגת כאן אינה יעילה, כמוה כקליטה של משתנים בודדים:

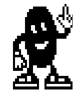
```
Readln (Tab[1]);
Readln (Tab[2]);
Readln (Tab[3]);
  .....
```

כדי לייעל את התוכנית, נשתמש במשתנה אשר מצביע כל פעם על איבר אחר בתוך המערך.

התוכנית הבאה מגדירה מערך בן עשרה תאים מסוג שלמים, וקולטת לתוכו עשרה ציונים של תלמידים:

```
1)   program  Klita1;
2)   var
3)       Index:         Integer;
4)       Marks:         array [1..10]  of  Integer;
5)
6)   begin
7)       for  Index := 1  to  10  do
8)           begin
9)               Writeln (Index, ' הכנס ציון מספר');
10)              Readln (Marks[Index]);
11)          end;
12)  end.
```

התוכנית מגדירה שני משתנים: בשורה 3 מוגדר משתנה מסוג שלם, ובשורה 4 מוגדר משתנה נוסף מסוג מערך, שבו עשרה תאים. בשורה 8 הוצהרה לולאת for שבה המשתנה Index משמש כמצביע על האיבר הראשון, השני, השלישי וכו', בתוך המערך. בשורה העשירית מתבצעת קליטת נתון. נשים לב כי המחשב מתייחס אל הביטוי Marks[Index] כאל משתנה מסוג שלם רגיל. כלומר, **כל איבר בודד בתוך מערך נחשב כמשתנה רגיל לכל דבר!**



הערה: *במקרים מסוימים, כאשר משתמשים בביטוי כמו Marks[Index], אין המחשב יכול לבקר תמיד את פעולותיו, מכיוון שהמצביע (במקרה שלנו Index) הוא משתנה. יכול להיווצר מצב, שבו המצביע יחרוג מן התחום שהוגדר למערך מסוים. תקלה מסוג זה מתבטאת בדרך כלל בהרס נתונים אחרים בתוכנית, או ל"נפילת" התוכנית ולכן יש לשים לב לעניין זה.*

מן הראוי להעיר שיש מהדרים שבונים מנגנון אבטחה שפועל בזמן ריצת התוכנית, ומאתר מצבים שבהם המצביע "חורג" מתחום המותר.

ניהול מערכים

עד כה עסקנו בניהול הבסיסי של מערכים. האופן שבו למדנו על מבנה המערך, וצורת השימוש בו נועדה לצורכי הדגמה והסבר בלבד. למעשה, כאשר כותבים תוכנית הכוללת מערכים, נהוג בדרך כלל לעשות כמה דברים יחד, אם ניתן מבחינת פסקל, ואם התוכנית מאפשרת זאת. לדוגמה, אם היה עלינו לכתוב תוכנית שמנהלת ציוני תלמידים בשתי כיתות שונות אנו יכולים להקים שני מערכים נפרדים, כי כך נוח יותר למשתמש. לעומת זה, כדי לכתוב תוכנית העוקבת אחר שינויי מזג האוויר שלוש פעמים ביום, עדיף לקרוא נתונים אל שלושה מערכים בו-זמנית. הדוגמאות הבאות ממחישות גישה זו.

קריאת נתונים נפרדת לכל מערך:

```
program Grades;
var
   I:           Integer;
   Class1:        array [1..10]  of  Integer;
   Class2:        array [1..10]  of  Integer;

begin
   Writeln ('הכנס ציוני כיתה ראשונה:');

   for  I := 1  to  10  do
      begin
         Writeln (I, ' :ציון מספר');
         Readln (Class[I]);
      end;

   Writeln  ('הכנס ציוני כיתה שנייה:');
   for  I := 1  to  10  do
      begin
         Writeln (I, ':ציון מספר');
         Readln (Class2[I]);
      end;
end.
```

בדוגמה זו אנו קולטים 10 ציונים המייצגים את ציוני הכיתה הראשונה, ולאחר מכן עוד 10 ציונים המייצגים את ציוני הכיתה השנייה. כפי שהוזכר בתחילת סעיף זה, בחרנו לקרוא את ציוני שתי הכיתות בנפרד, מטעמי נוחות למשתמש בתוכנה.

הדוגמה הבאה מדגימה **קריאת מספר ערכים בפעולה אחת** והכנסתם **לשלושה מערכים שונים** בו-זמנית.

```
program MultipleRead;
var
   I:          Integer;
   Check1:        array [1..10]  of   Real;
   Check2:        array [1..10]  of   Real;
   Check3:        array [1..10]  of   Real;

begin
   for  I := 1  to  10  do
      begin
         Writeln ('הכנס ערכי טמפרטורה ע"פ התבנית: בוקר, צהרים, ערב       ');
         Readln (Check3[I], Check2[I], Check1[I]);
      end;
end.
```

בשיטה זו אנו מנצלים את העובדה שאורכם של שלושת המערכים שווה, וקולטים לתוכם באופן מקביל, נתון אחד לכל מערך.

שילוב קבועים בהגדרת המערך

נתבונן בדוגמה הבאה:

```
program Demo;
var
   I:          Integer;
   List:        array [1..10]  of  Integer;

begin
   for  I := 1  to  10  do
       Readln (List [I]);

   for  I := 1  to  10  do
      Writeln (List [I]);
end.
```

לפנינו תוכנית פשוטה המדגימה חיסרון בולט בניהול המערך. אם נבחן את התוכנית לפרטיה, נמצא שאכן אין בה כל שגיאה או תקלה, אך לא בזאת מדובר. ניתן להבחין ממבט ראשון כי יש דבר משותף הן להגדרת המערך, והן לשימוש בו, או ה"סריקה" של איבריו.

אנו רואים כי תחום המערך נע בין 1 ל-10, וכך גם סריקתו שנעה בין 1 ל-10. נחשוב כעת מה היה עלינו לעשות אם הדרישות היו משתנות. ובמילים אחרות, מה עלינו לעשות כדי שהגדרת המערך תהיה בין 1 ל-12 או בין 3- ל-8. בתוכנית שבדוגמה היינו צריכים לשנות שלושה או אפילו שישה ערכים שונים ובתוכניות ארוכות יותר - אף עשרות ערכים.

שפת פסקל היא שפה מתוחכמת ויש בה כלים כדי להקל עלינו בתחום זה. ניתן לשלב בין ערכים קבועים למבנה הגדרת התוכנית. כיצד נעשה דבר זה? נתבונן בתוכנית שלפנינו:

```
program ConstLimit;
const
   ListLength  =  8;

var
   List:         array [1..ListLength]  of  Integer;

begin
   for  I := 1  to  ListLength  do
      Readln (List [I]);

   for  I := 1  to  ListLength  do
      Readln (List [I]);
end.
```

מתוך התוכנית הזו ניתן להבין את הרעיון. הגדרתה כוללת קבוע בשם ListLength, שבעזרתו הגדרנו את התחום העליון של המערך List ל-8, ובסיכום: התחום הוא 1 עד 8.

הרעיון הוא, שכאשר נרצה לשנות את תחום המערך העליון (או התחתון, במקרה אחר), כל שעלינו לעשות הוא לשנות את הקבוע ListLength.

תרגילים

תרגיל 8.1

כתבו תוכנית המגדירה מערך בשם Values שבו 10 תאים מסוג שלמים. בשלב הראשון התוכנית מאפסת את כל התאים. בשלב השני היא קולטת מספר סידורי של תא ואת הערך שיש להציב בו.

(**רמז:** יש לבדוק אם המספר הסידורי של התא המבוקש הוא בטווח ההגדרה של המערך).

תרגיל 8.2

כתבו תוכנית המגדירה מערך בשם Values שבו 5 תאים שבהם נתונים מסוג ממשי (Real). תפקיד התוכנית לקלוט אל תוך חמשת התאים ערכים ממשיים, לחלק את כולם ב-100, ולהדפיס את תוכנם לאחר החילוק.

תרגיל 8.3

כתבו תוכנית הסורקת מערך בשם NUM, שבו 10 תאים מסוג שלם. תפקיד התוכנית למצוא כמה ערכים במערך גדולים מ-0, כמה קטנים מ-0 וכמה שווים ל-0. השלב האחרון הוא הדפסת התוצאה.

תרגיל 8.4

כתבו תוכנית הקולטת ערך מסוים ובודקת כמה פעמים הוא נמצא בתוך מערך בן 20 תאים מסוג שלם. התוכנית תדפיס את מספר האיברים המכילים את הערך או לחילופין, תדפיס הודעה מתאימה כאשר הערך לא נמצא כלל.

תרגיל 8.5

כתבו תוכנית הבודקת אם סכום הערכים של התאים הזוגיים שווה לסכום הערכים של התאים האי-זוגיים במערך. המערך בנוי מ-10 תאים מסוג שלמים. לכל אחד מן המקרים תודפס הודעה מתאימה.

תרגיל 8.6

כתבו תוכנית המגדירה מערך בן 10 תאים. התוכנית מקבלת ערך התחלתי, ועל פיו היא ממלאת את המערך באופן סדרתי עולה. לדוגמה, אם המשתמש הכניס כקלט את המספר "3", תוכן המערך ייראה כך (משמאל): 13 ... 8 ,7 ,6 ,5 ,4 ,3.

תרגיל 8.7

כתבו תוכנית הקולטת עשרה ערכים שלמים אל תוך מערך, ומדפיסה את סכומם.

תרגיל 8.8

כתבו תוכנית שקולטת עשרה ערכים שלמים אל תוך מערך. לאחר מכן היא מדפיסה את הערכים האלה מסוף המערך לתחילתו, כך שהמספר האחרון שהוכנס יודפס ראשון.

תרגיל 8.9

כתבו תוכנית הבודקת אם כל איברי המערך שווים. התוכנית תדפיס הודעה בהתאם למסקנות הבדיקה.

תרגיל 8.10

כתבו תוכנית הקולטת 10 ערכים שלמים אל תוך מערך. התוכנית תבדוק אם המערך מכיל את כל הערכים מ-1 עד 10, ותדפיס הודעה מתאימה.

תרגיל 8.11

כתבו תוכנית הקולטת 10 ערכים שלמים אל תוך מערך. התוכנית תדפיס את המספר השני בגודלו מבין הערכים האלה.

תרגיל 8.12

כתבו תוכנית הקולטת עשרה ערכים שלמים. תפקיד התוכנית למצוא אם יש ערך החוזר על עצמו יותר מפעם אחת, ולהודיע על כך.

תרגיל 8.13

כתבו תוכנית המגדירה שני מערכים בעלי אותו גודל וסוג משתנה ממשי (Real). התוכנית "תחבר" את הערכים באיברים מקבילים בשני המערכים (איבר מס' 1 בשני המערכים, איבר מס' 2 בשני המערכים וכך הלאה) ותציג את הסכומים האלה על המסך. ההודעה תציין גם את מספר האיבר.

פרק 9

מערכים דו-מימדיים

שימושים של מערכים דו-מימדיים

בפרק הקודם עסקנו במערכים חד-מימדיים. למדנו מהו המערך, כיצד מגדירים אותו וכיצד משתמשים בו. נתבונן שוב בדוגמה של הדוור המחלק דואר כאנלוגיה למחשב המטפל באיברי מערך, אך נערוך בה שינוי קטן. במקום בתים בודדים נציב בניינים, שבהם כידוע ישנם דיירים רבים. למשל, ברחוב יש שישה בניינים שבכל אחד מהם ארבעה דיירים. כדי לטפל במקרים מסוג זה צריך להגדיר לכל בניין מערך משלו, שבו כל איבר מייצג דירה אחת. נוכל לכתוב את ההגדרות בצורה זו:

```
var
   Binyan1:     array [1..4]  of  Integer;
   Binyan2:     array [1..4]  of  Integer;
   Binyan3:     array [1..4]  of  Integer;
   ....
```

המערך Binyan1 מייצג את הבניין הראשון, המערך Binyan2 מייצג את הבניין השני וכך הלאה. האיבר Binyan1[1] מיוחס לדירה הראשונה בבניין הראשון, האיבר Binyan1[2] מיוחס לדירה השנייה בבניין הראשון וכך הלאה. ובאופן דומה, Binyan3[2] מיוחס לדירה השנייה בבניין השלישי.

אנו יכולים להבחין כי שיטה זו אינה יעילה ואף מקשה על ההבנה. נוסף על כך, אם היינו רוצים למפות עיר שלמה היינו משקיעים את רוב המאמץ בהגדרת המערכים. כאן באה לעזרתנו שפת פסקל. השפה מספקת את היכולת לשלב מספר מערכים בתוך מערך אחד. מערך מסוג זה נקרא **מערך רב-מימדי**, ואופן ההגדרה שלו דומה לאופן ההגדרה של מערך פשוט (חד-מימדי).

בתור דוגמה, נגדיר מערך דו-מימדי (בעל שני מימדים) הכולל את כל הבניינים ודירותיהם גם יחד:

```
var
   Binyanim:        array [1..6, 1..4]  of  Integer;
```

התחום הראשון [1..6] מייצג את מספר הבניינים, והתחום השני [1..4] מייצג את מספר הדירות בכל בניין. יש לשים לב **לפסיק** המבדיל בין שני התחומים.

כאשר נרצה לפנות לבניין מספר 3, דירה מספר 1 נוכל לכתוב:

```
Binyanim[3,1] := ...  ;
```

כדי להבין טוב יותר את מבנה הטבלה, נתבונן בטבלה הדו-מימדית הבאה, המייצגת **מטריצה**:

	1	2
1		X
2		
3		

את המטריצה הזו ניתן להגדיר בשפת פסקל באופן הבא:

```
var
   Matrix:       array [1..3, 1..2]  of  Integer;
```

בטבלה זו יש שלוש שורות ושתי עמודות. בדוגמת הבניינים הקודמת יש 6 שורות (שורה לכל בניין) וארבע עמודות (עמודה לכל דירה בבניין).

וכיצד **נפנה לאיבר כלשהו בטבלה** הדו-מימדית הזו:



זכרו: *כדי לפנות לאיבר במערך צריך לציין את מיקומו על פי מספר השורה ומספר העמודה ("קואורדינטות"). לשם תזכורת, במערך חד-מימדי צריך לציין מספר שורה בלבד!*

בטבלה Matrix שבדוגמה, האיבר Matrix[1, 2] הינו התא המסומן ב-X: שורה 1, עמודה 2.

כאשר נרצה לפנות לבניין מספר 5, דירה מספר 3 נוכל לכתוב:

```
Binyanim[5, 3] := ...  ;
```

דוגמאות אלו התייחסו למערך דו-מימדי, אך שפת פסקל תומכת בהרבה יותר מכך. לדוגמה, אם רוצים למפות שטח בשלושה מימדים (X, Y, Z) ניתן להגדיר מערך באופן הבא:

```
var
   Area:        array [1..10, 1..10, 1..10] of Real;
```

למערך זה שלושה מימדים, שכל אחד מהם הוא בן 10 איברים. באותו אופן ניתן להגדיר מערכים גדולים יותר בני שישה, שבעה מימדים ויותר.

פעולות במערך דו-מימדי

פעולות במערכים נעשות באמצעות לולאות, כדי לפשט ולייעל את תהליכי העבודה. ניהול של מערך דו-מימדי, או רב-מימדי, והפעולות (או סריקות) המבוצעות בו, דומים במידה רבה לאלה שנעשים במערך חד-מימדי.

נבחן כעת תוכנית המתחלת תוכן של מערך Matrix בעל שני מימדים, שכל איבריו הם מסוג ממשי.

```
program ClearMatrix;
var
   V:         Real;
   A, B:      Integer;
   Matrix:    array [1..3, 0..5]  of  Real;

begin
   Writeln ('הכנס ערך לתיחול המערך:');
   Readln (V);
   for  A := 1  to  3  do
      for  B := 0  to  5  do
           Matrix[A, B] := V;
end.
```

מטריצות

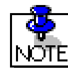
בתחום המחשבים ניתן להגדיר מטריצה כ**מערך דו-מימדי**. לדוגמה, מטריצה 4x5 היא מערך שבו יש ארבע שורות וחמש עמודות. אם כך, כאשר מדובר במטריצה ריבועית בגודל NxN (מספר השורות שווה למספר העמודות) מגדירים למעשה מערך דו-מימדי התואם את הגודל שנקבע. לדוגמה, אם מדובר במטריצה ריבועית שבה 3 שורות ו- 3 עמודות. ההגדרה של המערך נראית כך:

```
var
   Matrix:       array [1..3, 1..3]  of  Integer;
```

כעת ננסה לפתור תרגיל עם מטריצה ריבועית :

נתונה מטריצה ריבועית בגודל 4x4, ועלינו לבדוק אם היא **ריבוע קסם**. ריבוע זה הוא מטריצה, שסכום העמודות שלה שווה לסכום השורות, ושווה גם לסכום האלכסונים. ננסה ליצור תהליך, אשר יפשט את התוכנית. תחילה נחשב את סכום אחת השורות, ונניח כי זה הערך שיש בכל השורות, העמודות והאלכסונים. כעת נבדוק אם אמנם סכומם של כל אלה שווה. כמובן שניתן לעשות זאת בלוגיקה פשוטה של ניפוי : לבדוק שורה ראשונה, שנייה וכך הלאה ; ולאחר מכן את העמודה הראשונה, השנייה וכך הלאה ; ולבסוף - לבדוק את שני האלכסונים. תוכנית במבנה מסוג זה תכלול שורות רבות ותהיה מסורבלת. בעיה נוספת תיווצר, אם נרצה בעתיד לשנות את מבנה המטריצה. במקרה זה נאלץ לשכתב את כל התוכנית. לכן עלינו לפנות לפתרון יעיל יותר.

אחד הפתרונות הוא הוספה של מערך, שבו נאחסן את כל התשובות. לאחר מילוי המערך, נשתמש בשיטת הסריקה שלמדנו, ונבדוק אם כל הסכומים מתאימים.



זכרו: *המספרים משמאל למשפטי התוכנית מיועדים עבור ההסבר ואינם חלק מהתוכנית.*

```
1)    program  MagicalSquare;
2)    var
3)       A, B, Sum:     Integer;
4)       OK:            Boolean;
5)       Matrix:        array [1..4, 1..4]  of  Integer;
6)       Table:         array [1..10]  of  Integer;
7)                           { קטע ראשון }
8)    begin
9)       Sum := 0;
10)      OK := True;
11)      for A := 1  to  10  do  Table [A] := 0;
12)      for A := 1  to  4  do  Sum := Sum + Matrix [1, A];
13)                          { קטע שני }
14)      for  A := 1  to  4  do
15)          for  B := 1  to  4  do
16)              Table [A] := Table [A] + Matrix [A, B];
```

```
17)                    { קטע שלישי }
18)     for  A := 1  to  4  do
19)         for  B := 1  to  4  do
20)             Table [A + 4]:= Table [A + 4] + Matrix [B, A];
21)                    { קטע רביעי }
22)     for  A := 1  to  4  do
23)         Table [9] := Matrix [A, A];
24)                    { קטע חמישי }
25)     for  A := 1  to  4  do
26)         Table [10] := Matrix [5 - A, A];
27)
28)     for  A := 1  to  10  do
29)         if  Table [A] <> Sum  then  OK := False;
30)                    { קטע שישי }
31)     if  OK = True  then
32)         Writeln ('This is a Magic square !!!')
33)     else
34)         Writeln ('Not a magic square.');
35)   end.
```

ננתח את התוכנית, שלב אחר שלב ונעזר במספרים שמשמאל. בחלק הראשון של התוכנית העוסק בהגדרות מגדירים שלושה משתנים שלמים. שניים מהם (B, A) נועדו לסריקת המערכים, ותפקידו של המשתנה Sum להכיל את הסכום שנחשב בעמודות, בשורות ובאלכסונים. בשורה 4 מגדירים משתנה בוליאני שישמש בסוף התוכנית כ-"מסמן מצב". הוא יאמר לנו אם המטריצה היא ריבוע קסם, או לא. מגדירים את המטריצה עצמה בגודל 4x4, ומערך חד-מימדי נוסף שיכיל את כל התשובות. גודלו של מערך זה 10 תאים, מכיון שיש ארבע שורות, ארבע עמודות (ביחד זה שמונה) ושני אלכסונים, ויחד - עשר תשובות.

בקטע הראשון של התוכנית (שורות 9 עד 12) אנו מאתחלים את כל המשתנים. בקטע השני (שורות 14 עד 16) מחשבים כל שורה וכותבים את התוצאה בטבלה. תחילה נחשב את סכום השורה הראשונה (כאשר A=1) על ידי סריקה (המשתנה B מצביע כל פעם על תא אחר בשורה); לאחר מכן נחשב את סכום השורה השנייה (כאשר A=2) וכך הלאה.

בקטע השלישי של התוכנית מחשבים את העמודות, החל מהעמודה הראשונה (כאשר A=1, B מצביע על כל תא ותא בעמודה), העמודה השנייה (כאשר A=2) וכך הלאה. הביטוי Matrix [A+4] אומר שנתחיל למלא את התאים 5 עד 8, מכיון שתאים 1 עד 4 מלאים בסכומים של השורות. בקטע הרביעי (שורות 23,22 ו-26,25) מחשבים את סכום האלכסונים, כאשר A=1 אנו ניגשים לתא [1,1], וכאשר A=2 אנו ניגשים לתא [2,2] וכך הלאה.

הקטע החמישי הוא קטע הבדיקה. כאן אנו סורקים את כל התאים במערך TABLE ובוחנים אם כולם שווים. ברירת המחדל היא ריבוע קסם תקין. כלומר, בתחילת התוכנית (שורה 9) הגדרנו את המשתנה OK כ-True - הנחנו כי הריבוע הוא אכן ריבוע קסם. תפקיד קטע התוכנית הזה לנסות ולהפריך הנחה זו. אם במהלך הסריקה נמצאת שורה, עמודה או אלכסון שאינם תואמים לסכום של השאר - המשתנה OK יעודכן, ויושם בו הערך False, שיצביע על העובדה כי מטריצה זו אינה ריבוע קסם.

תפקיד החלק האחרון הוא להציג כפלט את תוכנו של משתנה OK. אם המשתנה מכיל את הערך True, כלומר הריבוע הוא ריבוע קסם - תודפס הודעה מתאימה לכך, ואם לא - תודפס הודעה אחרת.

קבועים ומערכים רב-מימדיים

בפרק הקודם, כאשר דנו במערכים פשוטים בעלי מימד אחד, הגדרנו כיצד משלבים את הגדרת המערך עם השמה של ערכים קבועים. באותו אופן ניתן להגדיר גם מערכים רב-מימדיים. נתבונן בדוגמה הבאה:

```
program ConstLimit;
const
   LowLimit     = 1;
   Limit1       = 5;
   Limit2       = 6;
   Limit3       = 4;

var
   Table:    array
      [LowLimit..Limit1,
       LowLimit..Limit2,
       LowLimit..Limit3]  of  Real;

begin
   ....
   ....
end.
```

בדוגמה זו הגדרנו מערך בעל שלושה מימדים: המימד הראשון מוגדר בתחום LowLimit עד Limit1 (1 עד 5); המימד השני מוגדר בתחום LowLimit עד Limit2 (1 עד 6); והמימד השלישי מוגדר בתחום LowLimit עד Limit3 (1 עד 4).

כעת נפתור **תרגיל** המשלב בהגדרת המערך ערכים קבועים.

כתבו תוכנית שבה יש מערך בגודל Nx8, כלומר N שורות (N הוא קבוע המוגדר בתחילת התוכנית) שבכל אחת מהן שמונה איברים מסוג בוליאני. ערך True מייצג "1", וערך False מייצג "0". תפקיד התוכנית להמיר כל שורה למחרוזת, ולהציגה על המסך.

```
program  BooleanTable;
const
   N = 5;

var
   Ch:        Char;
   A, B:      Integer;
   St:        string;
   Table:     array [1..N, 1..8] of  Boolean;

begin
   for  A := 1  to  N  do
       begin
           St := '';
           for  B := 1  to  8  do
                   begin
                       if  Table [A, B] = True  then
                         Ch := '1'
                       else
                         Ch := '0';
                         St := St + Ch;
                   end;
           Writeln  ('No. #', A, ' - ', St);
       end;
end.
```

ננתח את התוכנית ונלמד כיצד היא פועלת:

ההגדרות: בראש התוכנית ניצבת הגדרת הקבוע N. בחרנו במקרה את המספר 5. בתחום הגדרת המשתנים הגדרנו שני משתנים מסוג שלמים (A ו-B) לצורך סריקת המערך. בנוסף, הגדרנו משתנה מסוג תו לפישוט התוכנית, ואת תפקידו נבין במהלכה. המשתנה St מסוג מחרוזת הוגדר לצורך השמת הערכים המומרים, וגם את תפקידו נבין במהלך התוכנית. ולבסוף, הגדרנו מערך שיש בו N שורות (במקרה שלפנינו 5 שורות) ושמונה איברים בכל שורה.

גוף התוכנית: תפקיד הלולאה שבשורה 12 "להצביע" כל פעם על השורה שאנו סורקים, ואילו תפקיד הלולאה שבשורה 15 להצביע כל פעם על איבר אחר בשורה. פקודת ההתניה בשורה 17 בודקת מה תוכנו של התא הנוכחי, ובהתאם לכך מציבה "1" או "0" במשתנה Ch. תווים אלה מייצגים את הערכים True ו-False בהתאמה. לאחר ההמרה, התו מוסף למשתנה המחרוזת, כך שבסופה של הלולאה הפנימית, המשתנה St מכיל מחרוזת בעלת שמונה תווים. שורה 23 משמשת לפלט - הצגת מספר השורה והערך שהיא מייצגת.

תרגילים

תרגיל 9.1

כתבו תוכנית הכוללת מערך דו-מימדי בגודל 4x4, שבו האיברים מסוג בוליאני. התוכנית מקבלת כקלט ערכים מסוג שלם עבור כל איבר. אם הערך המוכנס הוא "0", יוצב באיבר המתאים "FALSE" ואם לא, יוצב בו "TRUE". בסיום הקליטה תדפיס התוכנית את תוכן התאים באופן הבא: אם האיבר הכיל "FALSE" יודפס "0", ואם הוא הכיל "TRUE" יודפס "1". יש להדפיס את המערך בצורת מטריצה ריבועית (באותו אופן שעורכים את לוח הכפל).

תרגיל 9.2

כתבו תוכנית הכוללת מערך דו-מימדי בגודל 4 שורות ו-3 עמודות, שאיבריו הם שלמים. התוכנית קולטת נתונים לכל התאים ומדפיסה סכום של כל שורה, וסכום של כל עמודה. יש להוסיף הודעה ברורה על מהות המספר שמודפס.

תרגיל 9.3

נתון מערך בגודל 5x5. כתבו תוכנית הממירה מערך זה למערך חד-מימדי בגודל 25 איברים (תאים). כל התאים הם מסוג ממשי.

תרגיל 9.4

כתבו תוכנית הקולטת מחרוזת של אותיות ומספרים. התוכנית תסרוק את המחרוזת ותמצא כמה פעמים מופיע בה כל תו. לדוגמה, אם המחרוזת הנקלטת נראית כך:

ZP2!WZ22

פלט התוכנית יהיה:

```
Z - 2
P - 1
2 - 3
! - 1
W - 1
```

תרגיל 9.5

כתבו תוכנית הקולטת ארבע מחרוזות שבכל אחת מהן עד חמישה תווים. התוכנית תקטע במידת הצורך את התווים המיותרים ותבצע המרה לכל מחרוזת, כדי שהמחרוזת הראשונה תוצב בשורה הראשונה במערך, המחרוזת השנייה תוצב בשורה השנייה וכך הלאה. המערך הוא דו-מימדי בגודל 4x5, וכל איבריו הם מסוג תו.

תרגיל 9.6

נתון מערך שבו ארבעים איברים מסוג תו. כתבו תוכנית הסורקת את המערך ומעתיקה אל מערך חדש את התווים, באופן שאף תו לא יחזור על עצמו יותר מפעם אחת.

תרגיל 9.7

נתון מערך בן עשרה איברים, שכל איבר בו הוא מסוג מחרוזת. באחת המחרוזות נמצאת המילה "START" ובאחרת (באיבר "גבוה" יותר) - המילה "STOP". כתבו תוכנית הסורקת את המערך, וכאשר היא נתקלת במילה "START" היא מתחילה להדפיס את כל המחרוזות באיברים הבאים, עד אשר היא מוצאת את המילה "STOP".

תרגיל 9.8

נתון מערך דו-מימדי בגודל 10x10 שכל ערכיו בתחום 1 עד 10. כתבו תוכנית הסורקת את המערך ומוצאת היכן נמצא הרצף הגדול ביותר של תווים זהים. הסריקה תיעשה הן לגבי שורות והן לגבי עמודות. לדוגמה, נתון מערך בגודל 5x5 ובו הנתונים הבאים:

```
2 2 4 5 6
1 3 4 7 1
1 4 8 7 1
9 9 8 7 9
3 4 5 3 2
```

פלט התוכנית יהיה:

מיקום: עמודה 4, החל משורה 2.
המספר: 7, מופיע 3 פעמים.

פרק 10

חיפוש

תהליכי חיפוש

בפרק זה נדון באחד המרכיבים החשובים של ניהול נתונים - החיפוש. **חיפוש** הינו סריקה של מערך נתונים כדי למצוא בו ערך כלשהו על פי קריטריון. למראית עין אין אנו צופים בעיות בתחום זה, אך אם נחקור לעומק, נמצא כי יש כאן מלכוד. המלכוד הוא, שככל שננהל טבלאות או קבצים גדולים יותר, כן ירבה הזמן אותו "יבזבז" המחשב כדי למצוא נתון מסוים. לכן, בפרק זה ננסה להציג דרכים שמטרתן למצוא את הנתון המבוקש בזמן הקצר ביותר.

בפרק זה נלמד על שני תהליכי חיפוש: הראשון הוא **חיפוש סדרתי**, שהוא הנפוץ ביותר, מכיון שהוא הפשוט ביותר. תהליך החיפוש השני הוא שיטת **החיפוש הבינארי**. שיטה זו מורכבת וגם מהירה יותר. בשני תהליכי חיפוש אלה ראוי לשים לב לתפקיד החשוב של המערך.

חיפוש סדרתי

יתרונות ה**חיפוש הסדרתי** הם הפשטות, והעובדה שהוא מתאים לכל מבני הנתונים. כדי להבין במה מדובר, נתבונן בקטע התוכנית הבא, שבו Key הוא הערך שאנו מחפשים בטבלה. כאשר מוצאים את הערך בטבלה, מעבירים למשתנה Position את מיקומו.

```
Position := -1;
for I := 1 to Size do
   if Key = Table[I] then Position := I;
```

קטע תוכנית זה מדגים את הרמה הנמוכה והפשוטה ביותר של תהליך החיפוש הסדרתי. תחילה אנו מציבים במשתנה השלם Position את הערך 1-, ומאוחר יותר נבין למה. כעת נסרוק את המערך בעזרת המשתנה I. הלולאה for "מניעה" את המשתנה I בין הערכים 1 ו-Size (קבוע שהוגדר בתחילת התוכנית).

כעת נעבור לתהליך הבדיקה: בעזרת המשתנה Key המשמש אותנו "כמפתח סריקה", שהוא הערך שאנו מחפשים, סורקים את המערך Table איבר אחר איבר. כאשר נמצא את מבוקשנו, המשתנה Position יקבל את מספר האיבר שבו נמצא הערך המבוקש. כעת אנו יכולים להבין מדוע תיחלנו את המשתנה Position לערך 1-. אם לא נמצא את הערך הרצוי, משתנה זה יישאר ללא שינוי (1-), ויסמן כי הערך שחיפשנו אינו נמצא במערך.

זו היתה השיטה הפשוטה ביותר לחיפוש סדרתי. אם נשים לב, נבחין כי שיטה זו איטית מאוד. הבעיה נובעת מהעובדה שגם לאחר שהערך שאנו מחפשים נמצא, הלולאה ממשיכה לרוץ וכך היא מבזבזת זמן יקר (יש לכך משמעות במערכים גדולים וכאשר מבצעים את החיפוש פעמים רבות!) לדוגמה, אם סורקים מערך בין מאה תאים, והערך שאנו מחפשים נמצא באיבר השלישי, הלולאה תמשיך לרוץ עד סופה למרות שמצאנו את מבוקשנו. לכן, נייעל את התהליך בצורה הבאה:

```
Position := -1;
I := 1;
while (I <= Size) and (Table[I] <> Key) do Inc  (I);
if  I <= Size  then  Position := I;
```

ננתח את משמעות הפקודות. בשתי השורות הראשונות תיחלנו את המשתנים Position ו-I. בשורה השלישית יצרנו לולאה שפועלת אך ורק אם שני התנאים הבאים מתקיימים:

- המשתנה I, המשמש אותנו כאינדקס, חייב להיות קטן מ-Size (גודלו של המערך). כאשר I יגיע לגודל זה, ניתן יהיה לומר כי כל המערך נסרק.
- התנאי השני להמשך פעילות הלולאה הוא שהתא הנוכחי שנבדק אינו הערך שאנו מחפשים.

במילים אחרות ניתן לומר, כי הלולאה תמשיך להתבצע כל עוד לא מצאנו את הערך שאנו מחפשים. היא עושה זאת על ידי העלאת ערך האינדקס I כדי שיוכל להצביע על האיבר הבא במערך, והשוואה של האיבר התורן במערך אל הערך המבוקש.

חיפוש בינארי

כדי להעריך את יעילות שיטת החיפוש הסדרתי, נתבונן בשני מקרים:

במקרה הראשון, הערך שאנו מחפשים נמצא באיבר הראשון, ואז הלולאה תתבצע פעם אחת בלבד. במקרה השני הערך שמחפשים נמצא באיבר האחרון במערך. במקרה זה, כאשר סורקים מערך שיש בו N איברים, הלולאה תתבצע עד מקסימום N פעמים. ללא מידע מוקדם על הערכים שבמערך ועל סדר הופעתם, ניתן יהיה לומר כי הממוצע של פעולות החיפוש הוא 2 / (N + 1).

לדוגמה, אם גודל (N) המערך הוא מאה בתים, מבחינה סטטיסטית נאלץ לסרוק חמישים תאים. מספר זה נחשב לקטן מאוד כאשר מדובר במחשב, אך מה יקרה כאשר יהיה נתון מערך בן 40,000 איברים, שנסרק כ-1000 פעמים בתוכנית? למצב כזה נדרשת שיטה טובה יותר.

שיטת החיפוש הבינארי מאפשרת להשיג יעילות רבה יותר בתהליכי חיפוש. בשיטה זו אנו מתבססים על העובדה שמערך הנתונים שלנו **ממוין**. זהו חסרונה הגדול של שיטה זו, אשר מחייב מיון מוקדם, או אחזקת איברי המערך בצורה ממוינת. עם זאת, יש יתרון במהירות החיפוש ביחס לשיטות חיפוש אחרות. כדי להבין את שיטת החיפוש הבינארי נתבונן בדוגמה הבאה:

מספר איבר:	11	10	9	8	7	6	5	4	3	2	1
ערך:	94	91	45	37	22	18	17	12	8	4	1

נניח שאנו מחפשים את המספר **37**, שנמצא באיבר השמיני במערך. ניגש תחילה לאיבר האמצעי, מספר 6 (קל לנו לעשות זאת כאשר מספר האיברים אי-זוגי). נשווה את הערך שבאיבר 6 אל הערך המבוקש 37. מכיון ש-37 יותר גדול, עלינו לחפש אותו בחלק השני, העליון, כי הערכים ממוינים בסדר עולה (זהו תנאי מוקדם להפעלת שיטת חיפוש זו).

נשתמש שוב באותה השיטה, ניגש לאיבר האמצעי שבין איבר מספר 6 לאיבר מספר 11. זהו תא 9 (על ידי חישוב פשוט ניתן למצוא, אותו כי 6+11)/2=8.5), ונעגל את התוצאה כלפי מעלה). כעת אנו משווים את הערך המבוקש 37 אל הערך 45 שנמצא באיבר 9. מכיון ש-45 גדול יותר מ-37, אנו צריכים "לרדת" לכיוון תחילת המערך.

עכשיו, ניגש לתא שנמצא בין המיקום הנוכחי לבין אמצע המערך, זהו תא 8. ואכן, בתא זה ישנו הערך שחיפשנו. אם נשווה שיטה זו אל שיטת החיפוש הסדרתי נמצא כי כאן היה עלינו לבצע שלוש השוואות, ואילו בשיטה הראשונה היה עלינו לבצע שמונה השוואות (בממוצע - 6).

איך פועלת שיטת החיפוש הבינארי (אנו מכירים אותה בשם "שיטה לתפיסת אריה במדבר"...)?

לפי שיטה זו לוקחים את טווח איברי המערך, מחלקים אותו לשניים, ומשווים את האיבר האמצעי לערך שמחפשים. מכיון שהמערך ממוין (לדוגמה, בסדר עולה) נוכל לדעת על פי הערך שבאיבר האמצעי אם הערך שאנו מחפשים נמצא בחצי העליון או בחצי התחתון של המערך. לאחר שמצאנו באיזה משני חצאי המערך נמצא הערך המבוקש, אנו משתמשים שוב באותה שיטה, וחוצים שוב את הטווח החדש (במקרה זה, חציו של המערך) לשניים. כך מתמשך התהליך עד אשר אנו מוצאים את הערך המבוקש, או לחילופין - אין אנו יכולים לחצות שוב את הקטע לשניים, והערך המבוקש לא נמצא.

משניתחנו את השיטה, נוכל לכתוב את התוכנית שמבצעת את החיפוש:

```
program BinarySearch;
const
   Size = 20;

var
   L, H, I, Key:  Integer;
   Table:         array [1..Size]  of  Integer;

begin
   Key := 1;

   L := 1;
   H := Size;
   repeat
          I := (L + H) div 2;
          if  Table[I] < Key  then  L := I;
          if  Table[I] > Key  then  H := I;
   until (Table[I] = Key) or (L = H);
end.
```

ננתח את מהלך התוכנית:

תחילה אנו מגדירים בשורה 2 את Size המייצג את גודל המערך. בשורה 6 מגדירים ארבעה משתנים: L המייצג את הגבול התחתון של הקטע הנבדק, H המייצג את הגבול העליון של הקטע הנבדק, I שישמש כאינדקס, ו-Key המייצג את הערך שמחפשים. נשתמש במערך Table לצורכי הדגמה.

בשורות 12 ו-13 אנו מאתחלים את המשתנים L ו-H, שיצביעו על תחילת המערך ועל סופו. כעת אנו פותחים בלולאה (שורה 14) אשר תסתיים כאשר נמצא את הערך הרצוי, או שתהליך הסריקה יסתיים ללא מציאת הערך המבוקש (שורה 18). בשורה 15 אנו מחשבים את המספר הסידורי של האיבר האמצעי בקטע המערך שאנו נמצאים בו (בתחילה זהו כל הטווח). בשורות 16 ו-17 אנו בודקים אם צריך לפנות לחצי העליון או התחתון שלו.

כך חוזר התהליך עד אשר נמצא את הערך המבוקש, או עד אשר קטע המערך שבו אנו מחפשים יתקצר לאיבר בודד (התחום העליון יהיה שווה לתחום התחתון L=H).

יישום של תהליכי חיפוש

כל אשר למדנו בפרק זה הוא שיטות לחיפוש. אך עדיין לא למדנו מהו השימוש שנעשה בהן. חיפוש במערך מיועד להשיג מידע רצוי כלשהו. לרוב, **קריטריון** או **תנאי החיפוש** מורכב ממספר תנאים, כפי שנראה בהמשך. חיפוש זה הינו **חיפוש על פי חתך**. לדוגמה, נוכל לומר למחשב להציג את כל שמות האנשים העובדים במפעל מסוים שגילם מעל 30, ושנות הוותק שלהם בעבודה לפחות 5. זהו הקריטריון, או **חתך של חיפוש**.

הקריטריון שהצגנו מכיל שני תנאים: התנאי הראשון הוא שכל האנשים שיוצגו ברשימה יהיו מעל גיל שלושים, והתנאי השני הוא שלכל אחד מהם יהיה לפחות ותק של חמש שנים. כיצד עושים זאת?

נתבונן בדוגמה הבאה:

```
program  ListNames;
type
    WorkerRec = record
        Name:      string;
        Age:       Integer;
        Vetek:     Integer;
    end;

var
    I:          Integer;
    List:       array [1..100]  of  WorkerRec;

begin
    for  I := 1  to 100 do
        begin
            if (List[I].Age > 30) and (List[I].Vetek >= 5) then
            begin
                Writeln  ('Name:  ', List[I].Name);
                Writeln  ('Age:   ', List[I].Age);
                Writeln  ('Vetek: ', List[I].Vetek);
            end;
        end;
end.
```

בתוכנית זו הגדרנו רשומה בשם WorkerRec המכילה את פרטי העובד: שם, גיל, ותק. הגדרנו משתנה שלם (I) אשר ישמש לסריקת המערך List הכולל עד מאה נתוני עובדים. כעת אנו מתחילים בסריקה (הלולאה for). בשורת ההתניה ... if ... then אנו שואלים שתי שאלות: "האם גיל העובד גדול מ-30" ו-"האם ותק העובד לפחות 5 שנים". אם התשובה לשתי שאלות אלו היא חיובית ("כן") - נוכל להדפיס את פרטי העובד.

שיטת החיפוש הבינארי אינה מתאימה במקרה זה בגלל סדר הנתונים.

תרגילים

תרגיל 10.1

כתבו תוכנית הקולטת עשר מחרוזות לתוך מערך, ולאחר מכן מחפשת בהן את המחרוזת "OK". אם המחרוזת נמצאה, התוכנית תדפיס את ההודעה "OK", ואם לא, היא תדפיס את ההודעה "NOT OK".

תרגיל 10.2

כתבו קטע תוכנית המבצע את החיפוש שבתרגיל 10.1 בעזרת הלולאה repeat ... until ....

תרגיל 10.3

כתבו תוכנית המגדירה מערך ובו תשעה איברים מסוג שלם. המערך מכיל את הנתונים הבאים כשהם ממוינים:

1, 6, 9, 11, 17, 18, 22, 24, 26

על התוכנית לחפש את הערך 22 בשתי הדרכים שלמדנו.

התוכנית צריכה גם למנות את מספר הצעדים שנדרשו לה בכל שיטה כדי למצוא את הערך המבוקש. את תוצאות הבדיקה יש להדפיס בסיום ההרצה.

תרגיל 10.4

כתבו תוכנית המגדירה רשומה בשם Student המכילה את שם התלמיד (מחרוזת עד 20 תווים), וממוצע הציונים שלו (משתנה ממשי).

התוכנית תגדיר שני מערכים מסוג הרשומה Student, בני עשרים איברים כל אחד. תפקיד התוכנית לזהות את כל התלמידים (במערך הראשון) אשר ממוצע ציוניהם גדול מ-80 ולהעתיקם למערך השני.

פרק 11

מיון ומיזוג מערכים

מיון מערכים

הקדמה

בפרקים קודמים למדנו על אופן הגדרת מערכים והדרכים לניהולם. גורם חשוב מאוד בתחום זה שעדיין לא נלמד הוא ה**מיון**. המיון הוא דרך לסדר נתונים במערך. לדוגמה, אם רוצים למיין קבוצת מספרים במערך, עלינו לסדר אותם לפי סדר מספרי. כאשר רוצים למיין שמות, אנו רוצים לסדר אותם לפי סדר אלפביתי. **סדר** זה יכול להיות **עולה** (**ascending**), מ-**0** עד **9** במיון מספרים, או מ-**א** (**A**) עד **ת** (**Z**); והוא יכול להיות ב**סדר יורד** (**descending**), מ-**9** עד **0** במיון מספרים או מ-**ת** (**Z**) עד **א** (**A**) במיון אלפביתי.

נסכם את מושגי היסוד בתחום זה:

- **מיון** - ארגון נתונים על פי סדר מסוים.
- **מפתח מיון** - גורם המגדיר על פי אילו קריטריונים יש לבצע מיון. לדוגמה, מיון של קבוצת נתונים נעשה על פי התו האחרון שלהם בלבד.
- **מיון עולה** - סידור נתונים, כך שהערך הקטן ביותר מופיע בראש הרשימה והגבוה יותר - בסופה. כלומר, סדר הנתונים הוא מן הנמוך לגבוה.
- **מיון יורד** - מיון נתונים, כך שהערך הגדול ביותר יהיה בראש הרשימה, והערך הקטן ביותר יהיה בסופה.

מיון שכנים

שיטת **מיון שכנים** היא שיטה שמשמשת אותנו גם בתהליכים יומיומיים, על אף שאין אנו מבחינים בכך. תהליך המיון פשוט למדי: מתחילים באיבר הראשון ומשווים אותו אל האיברים שאחריו במערך. אם מוצאים איבר שקטן ממנו,

מחליפים ביניהם. כשמגיעים לאיבר האחרון מקבלים בראש המערך את האיבר שערכו הקטן ביותר. אחר כך משווים את האיבר השני במערך אל כל שאר האיברים שאחריו (חוץ מהאיבר הראשון, שכבר עבר את תהליך המיון). אם נמצא איבר קטן ממנו, מחליפים ביניהם. בסופו של שלב זה יהיה באיבר השני ערך שגדול מערך האיבר הראשון, אך קטן מכל שאר הערכים. כך נמשך תהליך המיון, ובכל פעם משווים איבר אחד אל שאר האיברים שעדיין לא עברו את תהליך המיון. בסיום מקבלים מערך שאיבריו ממוינים.

שים לב שקבענו **סדר מיון עולה**. בסדר מיון **יורד**, יש שוני קטן וחשוב. התוכל לומר מהו?

נתבונן בדוגמת התוכנית כדי להבין את התהליך:

```
1)  program  Sort1;
2)  var
3)     A, B, Temp:    Integer;
4)     Table:         array [1..8]  of  Integer;
5)
6)  begin
7)     for  A := 1  to  7  do
8)         for  B := A + 1  to  8  do
9)             if  Table[B] < Table[A]  then
10)                begin
11)                    Temp := Table[A];
12)                    Table[A] := Table[B];
13)                    Table[B] := Temp;
14)                end;
15) end.
```

ננתח את התוכנית:

בשורה 3 אנו מגדירים שני משתנים (A, B) לצורך סריקת המערך, ומשתנה עזר (Temp) שיאפשר להחליף שני איברים. נחזור עתה אל תהליך המיון. אמרנו, כי משווים את האיבר הראשון לכל שאר האיברים. אם נתבונן בשורות 7 ו-8 כאשר A=1, נראה כי B נע בין 2 ל-8. כעת יש לנו שני משתנים, אשר אחד מהם מצביע על האיבר ראשון, והשני מצביע בתור על שאר האיברים. בסיום הלולאה השנייה (שורה 8 עד 14) המשתנה A מתקדם לערך הבא - 2, וכל הלולאה חוזרת על עצמה שוב, עד אשר A מגיע לאיבר שלפני האחרון.

אנו יכולים לשאול את עצמנו, מדוע אין A מגיע עד האיבר האחרון?

התשובה פשוטה - חיסכון בזמן הפעולה של המחשב. אם A יגיע לתא האחרון נשווה אותו אל עצמו, והתוצאה תהיה שוויון, כי מדובר באותו איבר. כך גם לגבי המשתנה B, הוא אינו מצביע על התא שהמשתנה A מצביע עליו.

מיון זה נקרא **מיון שכנים**, מפני שכל פעם נערכת השוואה בין שני איברים שכנים שונים.

מיון בועות

אחת השיטות הנפוצות למיון נתונים היא **מיון בועות**. כדי להדגים שיטה זו נאמר כי יש לנו מערך חד-מימדי בשם X ובו שמונה איברים מסוג שלם. המערך מכיל את הערכים הבאים (קרא משמאל לימין):

25, 57, 48, 37, 12, 92, 86, 33

השיטה עליה מתבסס תהליך מיון בועות: כל פעם משווים שני תאים צמודים זה לזה. אם התא הראשון מבין השניים גדול יותר - מחליפים ביניהם. לדוגמה:

X[1] עם X[2] (25 עם 57) - אין החלפה.
X[2] עם X[3] (57 עם 48) - יש החלפה. מעתה X[3] יהיה 57 ו-X[2] יהיה 48.
X[3] עם X[4] (57 עם 37) - יש החלפה. מעתה X[4] יהיה 57 ו-X[3] יהיה 37.
X[4] עם X[5] (57 עם 12) - יש החלפה. מעתה X[5] יהיה 57 ו-X[4] יהיה 12.
X[5] עם X[6] (57 עם 92) - אין החלפה.
X[6] עם X[7] (92 עם 86) - יש החלפה. מעתה X[7] יהיה 92 ו-X[6] יהיה 86.
X[7] עם X[8] (92 עם 33) - יש החלפה. מעתה X[8] יהיה 92 ו-X[7] יהיה 33.

כעת אנו רואים שבתא האחרון X[8] ישנו הערך הגדול ביותר - 92. כדי למיין את כל שאר איברי המערך אנו צריכים לחזור על הפעולה הזו עוד שבע פעמים מבחינה תיאורטית.

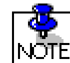
לפני שניגש לכתיבת תוכנית המדגימה מיון זה, נתעכב על נקודה מסוימת. ראינו כי לאחר המיון הראשון, נמצא בתא האחרון הערך הגדול ביותר. אם נמיין פעם נוספת, נראה שבתא שלפני האחרון נמצא הערך השני בגודלו במערך וכך הלאה. ניתן לנצל עובדה זו כדי לחסוך בזמן המיון, ולאחר כל שלב מיון להפחית מכמות התאים ש"ייכנסו" לשלב המיון הבא.

נתבונן בנקודה נוספת: אם לאחר סריקה של המערך לא התבצעו החלפות, ניתן לומר כי כל הערכים כבר ממוינים לפי הסדר, ולא יהיה צורך להמשיך ולסרוק את המערך. זהו למעשה סיום המיון.

התוכנית הבאה מדגימה מיון בועות:

```
program BubleSort;
var
   Switch:       Boolean;
   A, B, Temp:     Integer;
   Table:        array [1..8]  of  Integer;

begin
   A := 1;
   Switch := True;
   while  (A < 8)  and  (Switch = True)  do
      begin
         Switch := False;
         for  B := 1  to  8 - A  do
            if  Table[B + 1] < Table[B]  then
               begin
                  Temp := Table[B];
                  Table[B] := Table[B + 1];
                  Table[B + 1] := Temp;
                  Switch := True;
               end;
            A := A + 1;
      end;
end.
```



זכרו: *מיון זה נקרא* ***מיון בועות****, מכיון שבמהלכו המספרים הקטנים ("הקלים") "מבעבעים" כלפי מעלה, עד שהם תופסים את מקומם הסופי, לפי סדר המיון.*

יתרונות וחסרונות

ניתן לראות כי למיון בועות יש יתרון רב על פני מיון שכנים, בכך שניתן לחסוך בו בזמן הפעולה של המחשב. החיסרון, שאינו ברור למראית עין, הוא שבמקרים מסוימים קשה ליישם אותו. כאשר כותבים תוכנית, חייבים לבחור בשיטת המיון הטובה יותר והמתאימה לנתונים, מיון שכנים או מיון בועות.

יש שיטות מיון נוספות שלא נעסוק בהן כאן.

מיזוג מערכים

מיזוג מערכים אינה שיטת מיון נוספת, אלא ניצול התכונות של מערכים שכבר ממוינים. נניח לדוגמה, כי נתונים שני מערכים המכילים שמות ממוינים של תלמידי שתי כיתות שונות. לצורך כלשהו, למשל להכנת רשימת תלמידים לטיול, רוצים להשתמש **ברשימה אחת** שתכלול את שמות התלמידים משתי הכיתות.

השיטה הפשוטה והנראית לעין היא "לחבר" את שני המערכים ולאחר מכן למיין אותם. במקרה שלפנינו ודומים לו, אין דרישה ליעילות, ולכן נוכל לבחור גם בדרך זו שאינה הטובה ביותר.

במהלכי עיבוד שונים במחשב, שחוזרים על עצמם פעמים רבות, יש צורך לבצע פעולות מסוג זה ביעילות ולנצל את העובדה שהמערכים הנתונים **כבר ממוינים**. על כן, לא "נחבר" את המערכים ונמיין אותם כיחידה אחת, אלא **נמזג** את המערכים הממוינים. נוכל לחסוך זמן מחשב בשיטה זו כפי שנראה בהמשך.

בהנחה שלפנינו רצפי נתונים (מערכים) ממוינים, נוכל לפעול בשיטת **מיזוג מערכים.** תהליך זה "מחבר" שני מערכים תוך ניצול העובדה שהנתונים בכל מערך כבר עברו תהליכי מיון. מיזוג מערכים אפשרי ב**מערכים חד-מימדיים** בלבד.

תהליך מיזוג מערכים

תהליך המיזוג פשוט ומהיר, כפי שנראה בדוגמה הבאה:

```
program Merging;
var
    Pointer1:    Integer;
    Pointer2:    Integer;
    Pointer3:    Integer;
    Table1:      array [1..5]  of  Integer;
    Table2:      array [1..6]  of  Integer;
    Table3:      array [1..11]  of  Integer;

begin
    Pointer1 := 1;
    Pointer2 := 1;
    Pointer3 := 1;

```

```
    repeat
       if  Table1[Pointer1] <= Table2[Pointer2]       then
          begin
             Table3[Pointer3] := Table1[Pointer1];
             Pointer1 := Pointer1 + 1;
          end
       else
          begin
             Table3[Pointer3] := Table2[Pointer2];
             Pointer2 := Pointer2 + 1;
          end;
       Pointer3 := Pointer3 + 1;
    until  (Pointer1 > 5)  or  (Pointer2 > 6);

    if  Pointer1 < 5  then
          while  Pointer1 <= 5  do
          begin
             Table3[Pointer3] := Table1[Pointer1];
             Pointer1 := Pointer1 + 1;
             Pointer3 := Pointer3 + 1;
          end
    else
          while  Pointer2 <= 6  do
          begin
             Table3[Pointer3] := Table2[Pointer2];
             Pointer2 := Pointer2 + 1;
             Pointer3 := Pointer3 + 1;
          end;
end.
```

ניתוח התוכנית:

בחלק ה**הגדרה** של התוכנית הגדרנו שלושה מערכים: הראשון בן חמישה איברים, השני בין שישה איברים והאחרון בן אחד-עשר איברים, הוא מערך התוצאה שיוכל להכיל את שני המערכים הראשונים. בנוסף, הגדרנו שלושה מצביעים, אחד לכל מערך. ידוע לנו כי המערכים המקוריים ממוינים. לכן, כל אשר עלינו לעשות הוא להעתיק תא אחר תא לתוך מערך היעד (Table3) תוך שמירה על סדר המיון. הדרך הפשוטה ביותר לעשות זאת היא לבדוק בכל פעם באיזה מערך נמצא הערך הקטן ביותר, ולהעתיקו למערך היעד.

ניגש כעת ל**תהליך המיזוג** עצמו: בשורה 16 בודקים איזה משני הערכים קטן יותר, זה שנמצא באיבר הראשון במערך הראשון, או זה שבאיבר הראשון במערך השני. את הקטן מבין שניהם מעתיקים לאיבר הראשון במערך השלישי. כעת מעדכנים את המצביעים. אם העתקנו את הערך מהמערך הראשון, אנו מעדכנים את המצביע שלו, כדי שנוכל לבצע אותה פעולה על הערך הבא במערך. אם העתקנו את הערך מהמערך השני - נעדכן את המצביע שלו, ולא את המצביע של המערך הראשון. את המצביע של המערך השלישי אנו מגדילים ב-1 כדי שיצביע על המקום הבא במערך.

הלולאה שתחילתה בשורה 15 וסופה בשורה 27 מתנהלת עד אשר אחד המערכים הועתק במלואו למערך היעד. כעת, כל שנשאר לנו לעשות הוא להעתיק אל מערך היעד את שאר התאים מהמערך שלא "התרוקן".



הערה: *הפתרון שמוצג כאן למיזוג מערכים מניח שהמערכים ממוינים בסדר עולה (מהקטן לגדול) והתוצאה שתתקבל תהיה גם היא בסדר עולה. כאשר יש דרישה למיזוג מערכים הממוינים בסדר יורד, יש לשנות את סימני ההתניה בשורה 16.*

נחזור לדוגמה של רשימת שמות תלמידים בבית ספר, ונניח שיש בה 1000 שמות ממוינים. כעת, אנו רוצים להוסיף מספר שמות של תלמידים חדשים שהצטרפו במהלך השנה. פעולה זו אינה מצדיקה מיון שלם לאחר ההוספה, ולכן, מה נעשה?

כמובן: נמיין את הרשימה של התלמידים החדשים ונבחר בתהליך **מיזוג** כדי לקבל רשימה אחת שלמה ומעודכנת של תלמידי בית הספר.

מיזוג של יותר משני מערכים

דוגמת המיזוג בסעיף הקודם מתארת מיזוג של שני מערכים בלבד. מה יקרה אם עלינו למזג שלושה מערכים, ארבעה, ואף יותר? הפתרון אינו קשה. אפשר לבחור בדרך פשוטה של מיזוג שני מערכים בכל פעולה, עד אשר נקבל מערך בודד.

דרך מתוחכמת יותר, וכמובן שגם יעילה יותר, היא למזג במעבר אחד את הנתונים מכל המערכים. כלומר, קוראים את האיבר הראשון בכל מערך ובודקים מי מהם הקטן ביותר, ואותו מעבירים למערך היעד. מקדמים את המצביע באותו מערך לאיבר הבא ושוב בודקים מיהו האיבר היותר קטן ומעבירים אותו למערך היעד. כך ממשיכים בהעברת האיבר הקטן ביותר בכל סדרת השוואות. בשלב מסוים מסיימים לקרוא איברים ממערך כלשהו בשעה שבאחרים עדיין יש איברים. אם כן, ממשיכים למזג מספר קטן יותר של מערכים. לבסוף ייתכן שיישארו איברים במערך אחד שאותם נעתיק ישירות למערך היעד.

תרגילים

תרגיל 11.1

כתבו תוכנית הממיינת מערך חד-מימדי בגודל N בשיטת מיון שכנים. N הוא קבוע המוגדר בחלק ההגדרה של התוכנית. איברי המערך הם מסוג נתון ממשי.

תרגיל 11.2

כתבו תוכנית אשר תמזג שני מערכים חד-מימדיים שבכל אחד מהם עשרה איברים מסוג שלם. המערך הראשון ממוין בסדר עולה (מהקטן לגדול) והמערך השני ממוין בסדר יורד (מהגדול לקטן).

(**רמז**: כאשר שני מערכים ממוינים בסדר הפוך, ניתן להתחיל לקרוא בראשון מההתחלה לסוף, ובשני מהסוף להתחלה. דרך אחרת היא למיין את המערך ההפוך בסדר עולה ואחר כך למזג, אך האם תבחרו בדרך זו?).

תרגיל 11.3

כתבו תוכנית אשר ממזגת שלושה מערכים הממוינים בסדר יורד. כל מערך מכיל עשרה איברים מסוג שלם. אופן הביצוע נתון לבחירתכם.

תרגיל 11.4

נתונים שני מערכים חד-מימדיים המכילים שישה ציונים כל אחד (ערכים ממשיים). הציונים ממוינים בסדר יורד, אך עקב תקלה יש ביניהם גם ערכים שליליים. כתבו תוכנית אשר ממזגת את שני המערכים, תוך התעלמות מהערכים השליליים.

תרגיל 11.5

נתון מערך חד-מימדי שבו עשרה שמות תלמידים הממוינים בסדר עולה. כתבו תוכנית הקולטת שם נוסף, ומוסיפה אותו לרשימה תוך שמירה על סדר המיון.

תרגיל 11.6

נתונים שני מערכים בני חמישה תאים כל אחד. תוכנם של שני המערכים זהה (קראו משמאל):

14, 93, 44, 7, 16

כתבו תוכנית אשר תמיין את המערך הראשון בשיטת מיון שכנים ואת המערך השני - בשיטת מיון בועות. על התוכנית למנות כמה השוואות נעשו בכל שיטה ולהדפיס את התוצאות.

פרק 12

קבצים

מהו קובץ?

קובץ הוא **מבנה נתונים** על הדיסקט או על הדיסק המגנטי. אנו מכירים קבצים שונים המשמשים למטרות שונות. יש לנו קובצי תוכניות, קובצי מסמכים שכתבנו במעבד תמלילים, קובצי נתונים המכילים את מצב המלאי במחסן או את רשימת התלמידים ועוד. עלינו לברר לשם מה דרושים לנו הקבצים בדיסק או בדיסקט, ומה "חסר" לנו באחזקת כל אלה בזיכרון המחשב? מהי המשמעות האמיתית של **קובץ**, ומדוע אי אפשר לאחסן את כל התוכנות והמסמכים בזיכרון ולבחור בהם ככל שנרצה?

התשובה לכך ברורה. הזיכרון שאנו משתמשים בו הוא זיכרון נדיף (מתנדף, נעלם לאחר כיבוי המחשב). על כן אנו נאלצים **לשמור** את הנתונים והתוכניות בדיסקט או בדיסק הקשיח **ולטעון** אותם לזיכרון בעת הצורך. בנוסף לכך, הזיכרון שאנו משתמשים בו מוגבל בקיבולת שלו, ולכן לא ניתן לטעון את כל אוסף הנתונים לזיכרון לשם עיבוד (חשבו למשל, על קובץ תושבי המדינה, או על אלפי חשבונות הבנק). עלינו "לקרוא" מדי פעם חלקים של מאגר הנתונים שאנו צריכים לשימוש מיידי ולכתוב אותם לקובץ לאחר העדכון, כדי לפנות מקום לקבוצת הנתונים הבאה.

ניתן לומר, אם כן, ש**הקובץ** הוא כלי הקיבול של הנתונים. אליו אנו כותבים נתונים וממנו אנו קוראים נתונים הדרושים לנו לעיבוד. תוכנית היא מקרה פרטי, שבו המהדר יוצר עבורנו קובץ תוכנית ואנו קוראים אותו לצורך הפעלת התוכנית. אין בו תהליך של עדכון, כמו בקובץ נתונים.

הקבצים נכתבים בדיסקט או בדיסק קשיח. בהמשך נשתמש בביטוי "דיסק" בלבד. מן הראוי לומר שאפשר לכתוב קובץ גם לאמצעי אחר, כמו סרט מגנטי, CD-ROM ואמצעים אחרים, שלא נדון בהם כאן. אל המדפסת, למשל, אפשר לכתוב קובץ, אך אין לנו אפשרות לקרוא אותו ממנה. אנו גם כותבים אל המסך.

קבצים בשפת פסקל

שפת פסקל נותנת לנו כלים נוחים לניהול קבצים: יצירה, עדכון וגישה. כדי לאפשר נוחות זו יש צורך בכלים שונים לסוגי קבצים שונים, על פי שלושה קריטריונים: **קובצי טקסט**, **קבצים מובנים** (בעלי תבנית), **וקבצים בינאריים**. בפרק זה נלמד על שני הסוגים הראשונים.

קובצי טקסט (Text files)

קובצי טקסט מכילים מילים, משפטים, וגם מסמכים שהינם רצפים גדולים של מילים ומשפטים. קבצים אלה הם קבצים סדרתיים, שניתן לכתוב ולקרוא מהם בצורה **סדרתית** בלבד. לא ניתן, לדוגמה, לקרוא שורות אחדות בתחילת הקובץ, ולאחר מכן לדלג לאמצע להמשך הקריאה.

לטיפול בקובצי טקסט קיים סוג משתנה - **Text**. מגדירים אותו כשם שמגדירים משתנים אחרים:

```
var
   VarName:      Text;
```

VarName מייצג את שם המשתנה ו-Text מציין את סוג הנתון.

לדוגמה, כשאנו רוצים להגדיר משתנה בשם MyFile כמשתנה לניהול קובצי טקסט, נכתוב:

```
var
   MyFile:      Text;
```

כעת, משלמדנו כיצד מגדירים משתנה לניהול קובצי טקסט, נוכל ללמוד על השימוש בו.

הפקודה Assign (הכוונת שם קובץ)

כאשר רוצים לפנות לקובץ עלינו לציין תחילה את שמו. נעשה זאת באמצעות הפקודה **Assign** (שפירושה הכוונה) המייעדת משתנה מסוים לקובץ. משתנה זה יאפשר לנו לפנות אל נתוני הקובץ. תבנית הפקודה:

```
Assign (VarName, FileName);
```

VarName מציין את שם המשתנה, ו-**FileName** מציין את שם הקובץ שאנו רוצים לנהל. לדוגמה,

```
Assign (MyFile, 'FILE.DAT');
```

פקודה זו קובעת שהמשתנה MyFile יאפשר לפנות לנתוני קובץ FILE.DAT. כלומר, כאשר נשתמש במשתנה MyFile המחשב יידע שאנו רוצים לפנות לקובץ FILE.DAT.

הפקודה Reset (פתיחת קובץ לקריאה)

הפקודה **Assign** אינה מאפשרת קריאה או כתיבה לקובץ, אלא רק מייעדת משתנה לטיפול בקובץ מסוים. כאשר רוצים לפנות אל קובץ, עלינו "**לפתוח**" אותו תחילה. פתיחת הקובץ הינה פעולת הכנה של התוכנה לאיתור מקום הקובץ בדיסק וזיהוי "ראשו", כדי שאפשר יהיה לקרוא ממנו או לכתוב אליו.

בשפת פסקל יש פקודות שונות לפתיחת קובץ לקריאה, או לכתיבה.

פותחים קובץ לקריאה באמצעות הפקודה **Reset**. כך נכתוב את הפקודה:

```
Reset (VarName);
```

VarName מציין את משתנה הטקסט לניהול הקובץ. לדוגמה, השורות הבאות מורות למחשב לפתוח את הקובץ FILE.DAT לקריאה:

```
Assign (MyFile, 'FILE.DAT');
Reset  (MyFile);
```

אם ננסה לפתוח בעזרת הפקודה **Reset** קובץ שאינו קיים, המחשב יעצור את התוכנית ויודיע על שגיאה.

הפקודה Rewrite (יצירת קובץ לכתיבה)

הפקודה המשלימה לפקודת Reset היא הפקודה **Rewrite**. פקודה זו **מגדירה קובץ חדש ופותחת אותו לכתיבה** בלבד. אם קיים קובץ בשם זה, המחשב **ימחק** אותו וייצור קובץ חדש. כותבים את הפקודה Rewrite כשם שכותבים את הפקודה Reset. לדוגמה, השורות הבאות יוצרות קובץ חדש לכתיבה:

```
Assign (MyFile, 'FILE.DAT');
Rewrite (MyFile);
```



שימו לב: *הפקודה Rewrite יוצרת קובץ חדש, ואם יש קובץ בשם זה היא* ***מוחקת*** *אותו. ראו הוזהרתם!*

הפקודה **Append** (הוספת נתונים לקובץ)

בפעולות שונות עלינו "לעדכן" קובץ טקסט, כלומר **להוסיף** אליו נתונים בלי למחוק את הנתונים הנמצאים בו. לצורך כך משמשת הפקודה **Append**, שתפקידה לפתוח קובץ קיים לצורך עדכון. כלומר, לאתר את הקובץ בדיסק ולזהות את סופו, כדי שאפשר יהיה לכתוב בהמשכו נתונים חדשים.

שימו לב: *אנו עוסקים בקובץ טקסט, ופקודה זו מאפשרת אך ורק* ***להוסיף*** *נתונים חדשים לסוף הקובץ,* ***ואין כל אפשרות לשנות*** *בו דבר.*

כתיבה וקריאה מקובצי טקסט

כדי לפשט את תהליך הכתיבה לקבצים והקריאה מהם, הוחלט כי פקודות הפלט/קלט הבסיסיות המשמשות אותנו לכתיבה על המסך וקריאה מן המקלדת ישמשו אותנו גם לקריאה וכתיבה מקבצים. פקודות אלו הן כמובן Read ו-Readln, Write ו-Writeln.

השוני בכתיבת הפקודה הוא כמובן ציון העובדה שאנו רוצים לכתוב לקובץ או לקרוא ממנו, במקום לעשות זאת למסך ולמקלדת. את המסך והמקלדת אין צורך לציין במפורש בפקודה, כי הם אמצעי פלט וקלט סטנדרטיים, בהתאמה.

תבניות הפקודות לקריאה וכתיבה מקובץ:

```
Readln  (VarName, String);       קריאה מקובץ
Writeln (VarName, StringVar);    כתיבה לקובץ
```

VarName - שם משתנה אשר שויך לקובץ מסוים.

String או StringVar - משתנה שאליו נקרא את הנתונים, או שממנו נכתוב את הנתונים אל הקובץ.

לדוגמה, כדי ליצור קובץ חדש בשם DEMO.DAT ולכתוב לתוכו את המשפט "HELLO" נרשום את הפקודות האלו:

```
var
   MyFile:      Text;

begin
   Assign (MyFile, 'DEMO.DAT');
   Rewrite (MyFile);
   Writeln (MyFile, 'HELLO');
end.
```

באותו אופן, אם נרצה לקרוא שורה מקובץ AUTOEXEC.BAT שנמצא בספריה \:C, נכתוב את הפקודות האלו:

```
var
   MyFile:      Text;
   St:          string;

begin
   Assign (MyFile, 'C:\AUTOEXEC.BAT');
   Reset (MyFile);
   Readln (MyFile, St);
end.
```

כל התכונות והחוקים החלים על הפקודות Readln ו-Writeln נכונים גם לגבי קריאה מקבצים וכתיבה אליהם.

סגירת קבצים (File close)

כאשר אנו מסיימים את העבודה בקובץ עלינו ל"סגור" אותו. **סגירה** היא נוהל שבו התוכנית מודיעה למחשב שעליו לבצע פעולות מסוימות בקובץ, מכיון שאין אנו עומדים להשתמש בו יותר. **פעולת הסגירה חיונית**, במיוחד במצב של כתיבה.

לאחר הסגירה, המחשב אינו מאפשר לתוכנית לפנות שוב אל הקובץ, אלא רק אם תפתח אותו שוב. **במצב קריאה** - **קובץ קלט** - הקובץ נסגר ללא פעולות נוספות. לעומת זאת, אם הקובץ היה **במצב כתיבה** - **קובץ פלט** - סגירת הקובץ גוררת מספר פעולות: כתיבה לקובץ של הנתונים שטרם נכתבו מהמשתנה שהוגדר לקובץ, סימון סוף קובץ ועדכון טבלת הקבצים שבדיסק. בשלב מאוחר יותר נוכל לפתוח את הקובץ שוב לכל צורך שנרצה.

__שימו לב:__ אם לא סוגרים קובץ פלט הוא עלול להיות בלתי תקין, ולא יהיה ניתן לקריאה. חשוב לסגור את כל הקבצים לפני סיום התוכנית.

הפקודה לסגירת קובץ היא Close, והתבנית שלה:

```
Close (FileVar);
```

FileVar - המשתנה שהוגדר לקובץ.

לדוגמה, כדי לקרוא שורה מהקובץ AUTOEXEC.BAT בספריה \:C, לכתוב אותה על המסך ולסגור את הקובץ, נכתוב את הפקודות הבאות:

```
var
   MyFile:      Text;

begin
   Assign (MyFile, 'C:\AUTOEXEC.BAT');
   Reset (MyFile);
   Readln (MyFile, St);
   Writeln ('First line of C:\AUTOEXEC.BAT is ', St);
   Close (MyFile);
end.
```

הפונקציה Eof (זיהוי סוף קובץ)

אם נתבונן בדוגמה הקודמת נראה כי חסר בה מרכיב חשוב מאוד: אנו קוראים שורה בודדת ומסיימים. כדי לקרוא קובץ שלם עלינו ליצור לולאת קריאה, אך על מה תסתמך הלולאה? מה יסמן לה מתי להפסיק? איך נדע מהו **סוף הקובץ**?

לפתרון בעיה זו קיימת הפונקציה **Eof** (קיצור המילים **End Of File**). פונקציה זו מקבלת משתנה טקסט ומחזירה ערך בוליאני המציין אם הגענו לסוף הקובץ. לדוגמה, אם נרצה שהמשתנה SOF יציין את מצב הקובץ, נוכל לכתוב:

```
var
   SOF:         Boolean

begin
   ...
   SOF := Eof (MyFile);
   ...
end.
```

הגדרנו משתנה, אשר במהלכה של התוכנית יכיל את המצב שעליו מצביע המשתנה MyFile.

קבצים מבניים (Typed files)

קבצים מבניים הינם בעלי **מבנה קבוע ומוגדר**. כדי להבין זאת טוב יותר נגדיר שני מושגים בסיסיים:

שדה (Field) - סוג כלשהו של נתון. לדוגמה: מספר זהות, שם פרטי, כמות במלאי, גיל וכדי.

רשומה (Record) - קבוצה של שדות, או נתונים, מסוגים שונים, המתארים ישות כלשהי כמו אדם, פריט במלאי, חשבון בבנק וכוי. לדוגמה, שני השדות/נתונים שם תלמיד (מחרוזת) וציון (ערך ממשי) הם רשומה. גם קבוצת השדות תיאור שולחן (מחרוזת), גובה (ממשי), רוחב (ממשי) ואורך (ממשי) מהווה רשומה.

כעת נוכל להגדיר **קובץ מבני:** זהו קובץ אשר מכיל רשומות.

עד כה לא עסקנו במבנים מורכבים, אלא במבנים פשוטים כגון שלמים, ממשיים, מחרוזות ועוד. הגדרה של סוג משתנה שיוכל לכלול בתוכו סוגים שונים של משתנים בסיסיים נעשית באמצעות מילת ההגדרה **רשומה** (**Record**).

מבנה הרשומה

את מבנה הרשומות מגדירים באזור הגדרת המשתנים:

```
program Example;
type
   RecName = record
      VarName:      VarType;
      ...
   end;

var
   R:          MyRec;
   ...
```

תחום הגדרת הרשומה באזור הגדרת המשתנים מתחיל במילת המפתח **Type** (סוג**.** באזור זה אנו "יוצרים" משתנים חדשים המבוססים על משתנים מוכרים. כדי "ליצור" משתנה חדש נשתמש בתבנית

```
RecName = record
```

הגדרה זו מודיעה למחשב כי מעתה ועד הופעת המילה "end" מגדירים תוכן של רשומה. את השדות ברשומה מגדירים כמשתנים רגילים לכל דבר, לדוגמה:

```
type
   MarkRec = record
      Name:        string;
      Mark:       Integer;
   end;
```

ברשומה שבדוגמה הוגדרו שני שדות: "Name" מסוג מחרוזת, המכיל את שם התלמיד; ושדה "Mark" המייצג את הציון של התלמיד. לרשומה כולה קוראים "MarkRec". כאשר נרצה להגדיר משתנה חדש מסוג רשומה "MarkRec", עלינו לכתוב:

```
var
   M:          MarkRec;
```

כעת נשאלת השאלה, כיצד פונים לכל שדה בתוך הרשומה? כדי להבין זאת נתבונן בתוכנית הבאה:

```
program Example;
type
   MarkRec  =  record
      Name:        string;
      Mark:       Integer;
   end;

var
   M:          MarkRec;

begin
   Writeln ('Enter student name:');
   Readln (M.Name);
   Writeln ('Enter the mark:');
   Readln (M.Mark);
end.
```

לפי מבנה התוכנית רואים שניתן להתייחס לשדות אלה כאל משתנים רגילים, ואין כל שוני בינם לבין משתנה פשוט. ההבדל היחיד הוא תבנית השימוש: כותבים את שם המשתנה ואת שם השדה ומפרידים ביניהם בנקודה.

מכיון שניתן להתייחס לרשומות כסוגי משתנים רגילים כמו מחרוזת, שלם, ממשי ועוד, ניתן ליישם זאת גם על מערך:

```
const
   Limit = 30;

type
   MarkRec = record
      Name:        string;
      Mark:        Integer;
   end;

var
   Students:     array [1..Limit]  of  MarkRec;
```

בדוגמה זו הגדרנו מערך בן 30 איברים שכל אחד מהם הוא רשומה. מכאן גם ניתן להבין את ההבדל בין **תא** לבין **איבר**: **תא** הוא הגדרה למשתנה פשוט כגון משתנה שלם, ואילו **איבר** הוא הגדרה ל**בלוק** המכיל בתוכו כמה תאים.

כיצד כל זה מתקשר לקבצים?

אם נקשר בין שני הדברים שלמדנו - קבצים בעלי תבנית מוגדרת ורשומות - נוכל ליצור בתוכנית שלנו מערך מסוג חדש. מערך זה יוכל לקלוט נתונים בצורה מורכבת, לשמור אותם על הדיסק ולקרוא אותם מאוחר יותר כשנצטרך.

טיפול בקבצים מבניים

טיפול בקבצים מבניים דומה מאוד לטיפול בקובצי טקסט. תחילה נלמד כיצד מגדירים משתנה לטיפול בקבצים:

```
var
   VarName:      file of RecordName;
```

VarName הוא שם המשתנה. ההגדרה "file of RecordName" מציינת למחשב כי אנו רוצים ליצור משתנה לטיפול בקבצים מבניים, שהמבנה שלהם הוא זה המפורט בהגדרה של RecordName. נתבונן בקטע התוכנית הבאה:

```
type
   MarkRec = record
      Name:        string;
      Mark:        Real;
   end;

var
   MarkFile:     file of MarkRec;
```

קטע זה מגדיר משתנה בשם MarkFile המיועד לטפל בקבצים מבניים. כדי שהמחשב יוכל לדעת מהו מבנה הקובץ, כתבנו את שם הרשומה: MarkRec, במקרה זה.

לפני שנוכל להמשיך, עלינו לדעת מספר תכונות הקשורות לניהול קבצים מבניים.

התכונה הראשונה היא ש**קבצים מבניים אינם סדרתיים**. כתוצאה, ניתן לקרוא את הרשומות לפי הסדר שלהן בקובץ, אך גם אפשר "לנוע" בקובץ לפנים ולאחור ולקרוא כל רשומה רצויה.

תכונה חשובה נוספת היא, שלאחר פתיחת הקובץ אין צורך בפעולות נוספות לביצוע תהליכי קלט ופלט. לדוגמה, כדי לקרוא מקובץ טקסט עלינו לפתוח אותו בעזרת הפקודה Reset. כדי לכתוב בו חייבים להשתמש בפקודה Rewrite. בקבצים מבניים, לאחר שפתחנו את הקובץ בכל צורה שהיא, ניתן **גם לקרוא וגם לכתוב** ללא המגבלות הללו.

עתה נפנה לאופן השימוש בקבצים מבניים.

הפקודה Assign (הכוונת משתנה לשם קובץ)

השימוש בפקודה **Assign** להכוונת משתנה לשם קובץ נעשה כמו בקובצי טקסט. לדוגמה, בהסתמך על קטע התוכנית שזה עתה כתבנו, נוכל לייעד את המשתנה MarkFile לטיפול בקובץ MARKS.DAT על ידי המשפט הבא:

```
Assign (MarkFile, 'MARKS.DAT');
```

הפקודה Reset (פתיחת קובץ לקריאה וכתיבה)

הפקודה **Reset** פותחת קובץ לפעולות קלט ופלט. כאשר נעשה שימוש בפקודה זו המחשב יפתח קובץ שממנו נוכל גם לקרוא וגם לכתוב כל רשומה **בכל מקום** בקובץ. אם נבקש לפתוח קובץ שאינו קיים, המחשב יפסיק את מהלך התוכנית ויודיע על שגיאה.

הפקודה Rewrite (יצירת קובץ לכתיבה)

הפקודה **Rewrite** דומה מאוד לפקודה Reset, לאחר השימוש בה ניתנת האפשרות גם לכתוב וגם לקרוא מהקובץ. ההבדל היחיד בין שתי הפקודות הוא שבשימוש הפקודה Rewrite נוצר **קובץ חדש**, כלומר נקבל קובץ אשר יהיה ריק מנתונים, עד אשר נכתוב לתוכו.

שימו לב: *פעולה זו עלולה למחוק קובץ קיים. ראו הוזהרתם!*

הפקודה Close (סגירת קובץ)

תפקיד פקודה זו לסגור את הקובץ. פעולתה זהה לכל סוגי הקבצים.

קריאת נתונים (Read) וכתיבת נתונים (Write)

כדי להבין את הדרך שבה מתבצעת כתיבה לקובץ או קריאה ממנו, נתבונן בתוכנית הבאה, העוסקת בניהול מלאי פריטים. אנו קוראים נתונים מקובץ אחד ויוצרים קובץ חדש:

```
program  FileReadWrite;
type
  ItemRec = record
     Name:     string;
     Price:    Real;
  end;
var
  I:          ItemRec;
  F:          file of ItemRec;

begin
  { אתחול המשתנה I }
  I.Name := 'TELEVISION';
  I.Price := 2500.00;

  { יצירת קובץ חדש בשם PRICES.DAT וכתיבת נתונים לתוכו }
  Assign (F, 'PRICES.DAT');
  Rewrite (F);
  Write (F, I);
  Close (F);

  { פתיחה מחודשת של הקובץ החדש לצורך קריאת נתונים ממנו }
  Reset (F);
  Read (F, I);
  Close (F);
end.
```

ננתח את מבנה התוכנית:

בשורות 3 עד 6 אנו מגדירים רשומה חדשה בשם ItemRec, אשר מכילה שני שדות: השדה הראשון (Name) מסוג מחרוזת, מכיל את שם הפריט. השדה השני (Price) הוא מסוג ממשי, עבור מחיר הפריט. בשורה 8 מגדירים משתנה בשם I מסוג הרשומה שהגדרנו, ומתחלים אותו. בשורות 13 ו-14 מציבים בו נתונים: שם פריט (במקרה זה - TELEVISION) ומחיר הפריט (2500.00).

שימו לב: *כדי לפנות לשדה של הרשומה, אנו כותבים את שם הרשומה ואת שם השדה הרצוי לנו.*

כלומר, אם נכתוב פקודה כזו,

I.Name := 'TELEVISION'

הכוונה היא להכניס את הטקסט "TELEVISION" לתוך השדה/משתנה Name שברשומה I. באופן דומה אנו מתייחסים לשדה Price שברשומה זו.

לאחר שעשינו זאת, אנו רוצים לשמור את הנתונים בקובץ, ולכן בשורה 17 ציינו כי המשתנה F ייצג את הקובץ PRICES.DAT. לאחר זאת, יצרנו את הקובץ בעזרת הפקודה Rewrite שבשורה 18.

כעת נתבונן במבנה הפקודה Write שבשורה 19. קיימים שני פרמטרים: הראשון הוא משתנה לניהול הקובץ, והשני הוא המשתנה שאת תוכנו אנו רוצים לרשום לקובץ.

מבנה הפקודה **Write** הוא: שם המשתנה לניהול הקובץ ומשתנה נוסף המכיל את הנתונים. בהמשך נדון בהרחבה על מבנה הפקודה הזו.

בשורות 23 עד 25 אנו מבצעים את הפעולה ההפוכה: אנו פותחים את הקובץ על ידי הפקודה Reset, וקוראים ממנו את הרשומה שנכתבה לתוכו.

נוכל לסכם את מה שלמדנו בתוכנית זו: כאשר רוצים לקרוא מקובץ או לכתוב לתוכו בעזרת הפקודות Read ו-Write בהתאמה, עלינו לציין את שם המשתנה שמנהל את הקובץ ואת שם המשתנה שבתוכו יאוחסנו הנתונים, או שממנו הם ייכתבו לקובץ.

כל הקבצים המבניים אשר הצגנו בפרק זה נוצרו על פי **רשומות** שהגדרנו. קבצים מבניים ניתן גם להגדיר על בסיס **משתנים פשוטים** יותר. לדוגמה, ניתן להגדיר קובץ מבני המבוסס על ה**משתנה השלם** (**Integer**) ו**מחרוזת** (**String**) אשר קיבצנו כדי לענות על דרישה כלשהי.

נתבונן בדוגמה שלפנינו:

```
program  Example;
var
   F:          file of Integer;
   I:          Integer;
   Table:         array [1..10]  of  Integer;

begin
   Assign (F, 'NUMBERS.DAT');
   Rewrite (F);
   for  I := 1  to  10  do
      Write (F, Table[I]);
   Close (F);
end.
```

נסביר את התוכנית הזו. בחלק הראשון שלה הגדרנו שלושה משתנים: הראשון הוא משתנה לניהול קובץ מבני המבוסס על המשתנה השלם; המשתנה השני מסוג שלם הוגדר לצורך סריקת מערך; והמשתנה השלישי Table הוגדר כמערך חד-מימדי בגודל עשרה איברים מסוג שלם.

בתחילת התוכנית הכוונו את המשתנה F לניהול קובץ בשם NUMBERS.DAT. בשורה שלאחריו יצרנו את הקובץ בדיסק בעזרת הפקודה Rewrite. תפקיד הלולאה for לכתוב את תוכן תאי המערך לקובץ, תוך שימוש בשיטת הסריקה. בסיום התוכנית סוגרים את הקובץ בעזרת הפקודה Close.

הפונקציה Eof (סוף קובץ)

הפונקציה Eof מודיעה לתוכנית כאשר מגיעים לסוף הקובץ. משמעות הפקודה היא, שבעזרתה ניתן לבנות לולאות פשוטות לסריקת תוכן קבצים, ולדעת כאשר מגיעים לסוף הקובץ. לדוגמה, נניח כי NAMES.DAT הוא קובץ מבני המכיל מחרוזות. נוכל לקרוא את תוכנו לתוך מערך באמצעות לולאה פשוטה.

התוכנית הבאה מדגימה זאת:

```
program FileRead;
var
   F:           file of string;
   Counter:     Integer;
   Names:    array [1..100]  of  string;

begin
   Assign (F, 'NAMES.DAT');
   Reset (F);
   Counter := 0;
   while  Eof (F) = False  do
      begin
         Inc (Counter);
         Read (F, Names[Counter]);
      end;
   Close (F);
end.
```

בתוכנית זו הגדרנו משתנה לניהול קבצים מבניים המבוססים על מבנה המחרוזת. בנוסף הגדרנו גם מערך חד-מימדי הכולל 100 מחרוזות, ומשתנה בשם Counter אשר ימנה את מספר המחרוזות שקראנו מן הקובץ. כעת ניגש ישירות ללולאה while, שבה אנו משתמשים בפונקציה Eof בהתניה כזו:

כל עוד (while) לא הגענו לסוף הקובץ (Eof (F) = False),
תבצע (do) את הפקודות הבאות...

תוכן הלולאה: בכל פעם אנו מגדילים את המונה המשמש אותנו גם כאינדקס, וקוראים מחרוזת חדשה לתוך המערך. בסיום התוכנית סוגרים את הקובץ בעזרת הפקודה Close.

הפונקציה FileSize (מספר רשומות בקובץ)

פונקציה נוספת הבאה לעזרתנו בניהול קבצים מבניים היא **FileSize**. על פי שמה ניתן להבין שתפקידה לדווח על גודל הקובץ. נדייק ונאמר, כי פונקציה זו מדווחת על **מספר הרשומות** בקובץ. לדוגמה, אם מנהלים קובץ שמכיל נתונים על פריטים ומחיריהם, הפונקציה FileSize תוכל לדווח כמה פריטים (רשומות) מצויים בקובץ. דוגמה נוספת: כאשר נתון קובץ המכיל מחרוזות, הפונקציה FileSize תוכל לדווח כמה מחרוזות קיימות בו.

כעת ננסה לממש את הדוגמה האחרונה בצורה קצת שונה. במקום להשתמש בלולאה while והפונקציה Eof, נשתמש בלולאה for והפונקציה FileSize :

```
program  FileRead;
var
   F:          file of string;
   I:          Integer;
   Names:       array [1..100]  of  string;

begin
   Assign (F, 'NAMES.DAT');
   Reset  (F);
   for  I := 1  to  FileSize (F)  do
      Read  (F, Names[I]);
   Close (F);
end.
```

הפונקציה Seek (איתור נתון לפי מיקום)

אחד היתרונות הגדולים של הקובץ המבני הוא שניתן **לנוע בו מרשומה לרשומה**. נסביר ונדגים זאת בעזרת הפקודה **Seek**. נניח שנתון קובץ שבו חמש מחרוזות, ועלינו לשנות את המחרוזת השלישית. כיצד נעשה זאת? עלינו "לומר" למחשב לקרוא את הנתונים של המחרוזת המסוימת. זאת נעשה באמצעות הפקודה Seek, אך כיצד משתמשים בה? לפני שנשיב, נתבונן תחילה איך פונים לרשומה.



שימו לב: *אנו נוהגים לציין מספר סידורי מהערך 1. במחשב מתחילה הספירה מ-0. לכן, כאשר נרצה "לגשת" לנתון הראשון בקובץ, לקרוא או לכתוב אותו, נציין למחשב כי אנו מבקשים גישה לנתון "מספר 0". הנתון השני יהיה מספר 1, החמישי יהיה מספר 4 וכן הלאה.*

נחזור לדוגמה. ברצוננו לשנות את המחרוזת השלישית. מבחינת המחשב זוהי מחרוזת "מספר 2" בקובץ. כעת, כל אשר נשאר לנו לעשות הוא לפנות למחרוזת זו ולקרוא אותה. תבנית הפקודה Seek :

```
Seek (FileVar, Position);
```

FileVar - מייצג את המשתנה שיועד לקובץ שמטפלים בו.

Position - הוא המיקום בקובץ שאליו רוצים לגשת.

בדוגמה זו, הפקודה seek פונה אל המחרוזת הרצויה באמצעות הערך Position "המצביע" על מקומה הסידורי בקובץ (אשר מתחיל מ- 0).

כעת נכתוב תוכנית המציגה במסך את המחרוזת השלישית של הקובץ ששמו FILE.DAT :

```
program DisplayString;
var
   F:          file of string;
   St:         string;

begin
   Assign (F, 'FILE.DAT');
   Reset (F);
   Seek (F, 2);
   Read (F, St);
   Close (F);
   Writeln ('The 3rd string in the file is: ', St);
end.
```

הפונקציה FilePos (מיקום מצביע הקובץ)

בעזרת הפקודה Seek ניתן לנוע בתוך הקובץ אל מחרוזת רצויה. מכיון שתכונה זו קיימת עבור קבצים מבניים, חייבת להיות גם דרך אשר נוכל באמצעותה לדעת **היכן נמצא המצביע** של הקובץ. כלומר, אנו רוצים לדעת על איזו מחרוזת או רשומה מוצב המצביע. לשם כך נשתמש בפונקציה **FilePos** שתפקידה לדווח על כך. תבנית הפונקציה:

```
Position := FilePos (FileVar);
```

- **FileVar** - המשתנה לניהול הקובץ.
- **Position** - משתנה שלם שמקבל לתוכו את מספר המחרוזת או את מספר הרשומה שעליה מוצב המצביע.

כעת נכתוב תוכנית שמדפיסה את גודל הקובץ (מספר הרשומות שבו) על ידי שימוש בפקודה FilePos :

```
program CalcFileSize;
var
   F:       file of Integer;
   Size:    Integer;

begin
   Assign (F, 'FILE.DAT');
   Reset (F);
   Seek (F, FileSize(F));
```

```
10)      Size := FilePos (F);
11)      Writeln ('File size is:  ', Size);
12)      Close (F);
13)   end.
```

נבחן את התוכנית הזו:

בשורה 3 הגדרנו משתנה לניהול קובץ המכיל מספרים שלמים. בשורה 4 הגדרנו משתנה Size שתפקידו להכיל את גודל הקובץ. בשורה 7 אנו מכוונים את המשתנה F לניהול הקובץ FILE.DAT, ובשורה 8 פותחים את הקובץ.

שורה 9 היא שורת המפתח של תוכנית זו. בשורה זו אנו מורים למחשב לגשת למקום מסוים בקובץ, וכאן - אל סוף הקובץ, על פי הפונקציה FileSize שמחזירה את גודל הקובץ. כזכור, גודל הקובץ נמדד במספר מחרוזות, או רשומות שנמצאות בו. כל אשר נשאר לנו לעשות כעת הוא לקרוא את מיקום מצביע הקובץ ולהדפיס את ערכו. בשורה 10 אנו מציבים במשתנה Size את מיקום המצביע, ומדפיסים הודעה בשורה 11.

תרגילים

תרגיל 12.1

כתבו תוכנית הקוראת את תוכן הקובץ C:\AUTOEXEC.BAT ומדפיסה את כולו על המסך.

תרגיל 12.2

כתבו תוכנית הקולטת שישה מספרים שלמים לתוך מערך חד-מימדי וכותבת אותם לקובץ טקסט.

תרגיל 12.3

כתבו תוכנית הקוראת את כל הרשומות הזוגיות שבקובץ LIST.DAT לתוך מערך. הקובץ מכיל ערכים שלמים. אחר כך היא מדפיסה את מספר המשתנים שנטענו לתוך המערך.

תרגיל 12.4

כתבו תוכנית הקוראת מחרוזת ושומרת אותה בקובץ טקסט בשם LINE.TXT. תהליך זה יימשך עד אשר המחרוזת שתתקבל תהיה ריקה (המשתמש ילחץ Enter בלי להקיש תווים נוספים).

תרגיל 12.5

כתבו את חלק ההגדרה (var ו-type) בלבד של תוכנית הקוראת קובץ שמבנה הרשומות שלו הוא:

שם עובד	גיל	ותק בעבודה

תרגיל 12.6

כתבו תוכנית המנהלת קובץ מבני שהרשומה שלו כוללת מקום לשם פרטי, שם משפחה ומספר טלפון. התוכנית תקלוט נתונים אלה ותרשום אותם בקובץ PHONE.DAT בצורה סדרתית, עד אשר המשתמש יכניס את המחרוזת "000", אשר תציין לתוכנית לסיים את פעולתה.

אורך המחרוזת של השם הפרטי 15 תווים, אורך המחרוזת של שם המשפחה הוא 20 תווים, ואורך המחרוזת למספר הטלפון הוא 10 תווים.

תרגיל 12.7

בהסתמך על התרגיל הקודם, כתבו תוכנית אשר קוראת את כל הנתונים מהקובץ על פי הפירוט שניתן בשאלה הקודמת, ומדפיסה אותם על המסך.

תרגיל 12.8

כתבו תוכנית המקבלת שני שמות קבצים. התוכנית תקרא את הקובץ הראשון כקובץ טקסט ותעתיק אותו אל הקובץ השני.

תרגיל 12.9

נתונים שני קבצים מבניים (FILE1.DAT ו-FILE2.DAT) בגודל זהה המכילים ערכים שלמים. כתבו תוכנית אשר תעתיק מחרוזת מכל קובץ לסירוגין ותכתוב אותם לקובץ היעד FILE3.DAT.

לדוגמה, נקרא מחרוזת ראשונה מהקובץ הראשון, מחרוזת ראשונה מהקובץ השני, מחרוזת שנייה מהקובץ הראשון, מחרוזת שנייה מהקובץ השני, וכך הלאה.

מטלה נוספת: עכשיו כתבו תוכנית אחרת, שבה קוראים מהקובץ הראשון כל מחרוזת שמקומה בקובץ אי-זוגי וקוראים מהקובץ השני כל רשומה זוגית. את הרשומות שקוראים כותבים לקובץ שלישי.

נספח א

ערבול (HASHING)

הקדמה

למדנו עד כה על דרכים שונות לניהול נתונים, מיון וחיפוש. בשיטות חיפוש סדרתי וחיפוש בינארי אנו מסתמכים על העובדה שיש לבצע מספר מסוים של צעדים כדי למצוא את הנתון המבוקש, או לחילופין, להוכיח שהוא אינו נמצא במערך הנתונים שסרקנו.

עתה עלינו להשיב על שאלה בסיסית ונוקבת: האם נאלץ לבזבז את זמנו של המחשב על חיפוש נתונים שייתכן כי אינם קיימים כלל? האם יש שיטה "למצוא" נתון מבוקש ללא תהליך חיפוש? התשובה לכך היא **כן**! ניתן ליצור מצב, בו לא נאלץ להשתמש בתהליך החיפוש כדי לאתר ערך או נתון מבוקש בתוך מערך, או קובץ. לשם כך נשתמש בשיטות **ערבול** (**Hashing**), או **טחינה**, כפי שיש הנוהגים לכנות מושג זה בעברית.

נבחן את המצב הבא: בכיתה 40 תלמידים. לכל תלמיד יש מספר סידורי המיועד אך ורק לו. כאשר המורה החליטה לחשב ולשמור את ממוצעי ציוני התלמידים, היא בחרה לעשות זאת בעזרת מערך שהוגדר באופן הבא:

```
var
   Marks:      array [1..40] of Real;
```

בדרך זו, ממוצע הציונים של תלמיד שמספרו האישי 1 יישמר באיבר מספר 1 (Marks[1]) של המערך, וממוצע הציונים של התלמיד שמספרו האישי 2 יישמר באיבר מספר 2 (Marks[2]) וכך הלאה. כך אנו פותרים את בעיית החיפוש, מכיון שהמספר האישי של התלמיד משמש גם כמפתח חיפוש של הנתון, או הערך, השייך לו. לכאורה נראה שפתרנו את הבעיות כי שיטה זו נראית בטוחה וחסכונית, אולם אין זה כך. השיטה מתאימה אך ורק למצבים מסוימים, כפי שנראה מייד.

נבחן את המצב הבא: במפעל 150 עובדים. אנו רוצים לנהל את נתוניהם של העובדים בשיטה שנהגנו לגבי התלמידים, תוך שימור על תאימות עם יישומים אחרים העוסקים בנתוני העובדים. על כן נשתמש במספר תעודת הזהות של העובד כמפתח חיפוש, וכאן אנו נתקלים בבעיה: מספר תעודת הזהות מורכב מ-7 ספרות, דבר שידרוש מאיתנו להגדיר מערך שבו מיליוני איברים, בעוד שרק 150 מהם ינוצלו עבור 150 העובדים. בזבוז צפוי זה מחייב אותנו למצוא פתרון אחר. נוכל כבר עכשיו לומר, כי אין פתרון אידיאלי לבעיה מסוג זה, אולם יש דרכים המציעות לנו פתרונות מספקים וסבירים.

דוגמה לביצוע ערבול

לפנינו משימה: עלינו לכתוב תוכנית לבית ספר בו יש עד 1000 תלמידים. לכל תלמיד יש מספר תעודת זהות בן 7 ספרות. בדוגמה הקודמת קבענו כי השימוש במספר תעודת הזהות אינו פתרון טוב לבעיה. למרות זאת, נוכל להשתמש למשל, בשלוש הספרות האחרונות של תעודת הזהות כמפתח. לדוגמה, כאשר תלמיד יבקש את מאזן ציוניו, הוא יציין את מספר תעודת הזהות שלו. המחשב יבודד את שלוש הספרות האחרונות של המספר, ובעזרתן ישלוף את הנתונים של התלמיד. נציג זאת בעזרת אלגוריתם מילולי:

התחלה

הצגת הודעה המבקשת את מספר תעודת זהות של התלמיד

קליטת המספר לתוך המשתנה KEY.

H(KEY) = MOD(KEY, 1000)

הדפסת H(KEY)

סוף

ננתח את מהלך האלגוריתם:

הצגנו הודעה המבקשת את מספר תעודת הזהות ואחר כך קלטנו אותו לתוך המשתנה השלם KEY. בעזרת הפונקציה MOD חישבנו את השארית של חלוקת המספר KEY ב-1000. בעשותנו זאת בודדנו את שלוש הספרות האחרונות של מספר תעודת הזהות. כל שנשאר לנו לעשות הוא "לגשת" לאיבר במערך הציונים המכיל את ציוני התלמיד, וכמובן, בעזרת המפתח שחישבנו. הכלי שבעזרתו אנו מסיבים מפתח לכתובת נקרא **פונקציית ערבול** (**Hashing function**). בדוגמה זו הפעלנו את הפונקציה MOD בתפקיד של פונקציית ערבול. הטבלה הבאה מציגה את הפעולה שביצענו:

מספר תא	מספר ת.ז.
0	4567**000**
1	3912**001**
2	8764**002**
:	:
:	:
997	5643**997**
998	3765**998**
999	4923**999**

מכל מספר תעודת זהות אנו "מקצצים" את ארבע הספרות העליונות ומשתמשים בשלוש הספרות התחתונות (מימין) לצורך שליפת הנתון מתא מסוים.

לשיטה זו חיסרון גדול, שנסביר אותו על ידי שאלה:

האם שני מפתחות סידוריים, (למשל, KEY1 ו-KEY2)

שאינם שווים (KEY1 < > KEY2)

יכולים לגרום לפונקציית הערבול לדווח על **תוצאה שווה**

(כלומר, H(KEY1) = H(KEY2)).

התשובה היא - **כן!**

לדוגמה, שני מספרי תעודות הזהות 3941559 ו-1957559 ייצרו את אותה תוצאה של פונקציית הערבול שהשתמשנו בה קודם, מכיון ששלוש הספרות האחרונות שוות בשני מספרי זהות אלה. למצב בו הפונקציה מספקת תוצאה שווה למקרים שונים אנו קוראים **התנגשות - Collision**. ניתן להפחית את סיכויי ההתנגשות על ידי הגדלת טווח הערכים. כלומר, להשתמש למשל, בארבע הספרות של מספר הזהות במקום בשלוש. דבר זה מקטין את הסיכוי להתנגשות, אולם הוא ייאלץ אותנו להקצות שטח עבודה (איברים) פי 10 יותר גדול. וזהו חיסרון גדול מאוד של השיטה הזו. אף על פי כן, כאשר ניתן להשתמש בשיטה זו, רמת הטיפול בנתונים היא הטובה, המהירה והיעילה ביותר.

תרגיל

במפעל עשרה עובדים שלכל אחד מהם יש מספר עובד המייצג אותו. מספרי העובדים הם:

636, 975, 190, 436, 987, 533, 123, 879, 452, 567

רשימת עובדים זו מנוהלת בטבלה בת עשרה איברים בתחום 0..9.

חשב לפי פונקציית השארית HASH(KEY) = MOD(KEY, 10) את הכתובת של כל עובד.

מפתח	כתובת מחושבת
567	7
452	2
879	9
123	3
533	3
987	7
436	6
190	0
975	5
636	6

על-פי טבלה זו ניתן לראות כי מספר ההתנגשויות הוא שלוש (המספר 7 מופיע יותר מפעם אחת, המספר 3 מופיע יותר מפעם אחת והמספר 6 מופיע יותר מפעם אחת - סה"כ שלוש).

כדי לפתור את בעית ההתנגשות נגדיל את טווח הטבלה ל-100, כלומר תחום האיברים יהיה 0..99.

כעת נחשב את הכתובת על פי הפונקציה HASH(KEY) = MOD(KEY, 100). לפי פונקציה זו נקבל:

מפתח	כתובת מחושבת
567	67
452	52
879	79
123	23
533	33
987	87

436	36
190	90
975	75
636	36

כך הגדלנו את הטווח והקטנו את הסיכוי להתנגשות. על פי טבלה זו אנו רואים שישנה התנגשות אחת בלבד: המספר 36 מופיע יותר מפעם אחת.

לבעיה שהצגנו כאן אין פתרון טוב באמצעות פונקציית הערבול שהשתמשנו בה, כי גם הפעם נאלץ להכפיל את טווח הטבלה פי 10 כדי לקבל את התוצאה המבוקשת - אי התנגשות. דרך זו אינה מעשית, מכיון שעלינו להשתמש בעשרה איברים בלבד מתוך 1000, וזהו בזבוז רב. לשם כך משתמשים בפונקציות ערבול אחרות, יעילות יותר.

פונקציות ערבול נוספות

כפי שזה עתה ראינו, פונקציית ערבול שבה אנו משתמשים במספר ספרות מתוך ערך נתון (לדוגמה, מספר תעודת זהות) לוקה בחסר. לרוב, נוצר מספר גבוה של התנגשויות ביחס לכמות הנתונים. לשם כך עלינו לפנות לפתרונות חלופיים שינסו לתת תוצאות טובות יותר. תוצאת האלגוריתם שנפעיל צריכה "ליפול" בתחום המספרים שהקצינו (כמו לדוגמה, 10, 100 או 1000).

לפנינו מספר דוגמאות לפונקציות ערבול:

פונקציית ריבוע: בטכניקה זו אנו לוקחים את המפתח, מעלים אותו בריבוע ושולפים מתוכו שלוש ספרות. לדוגמה, כאשר נתון המפתח 4567 ונעלה אותו בריבוע, נקבל 2087489. כאשר הדרישה תהיה ליצור מפתח בן שלוש ספרות, נוכל לבחור את הערך 874 מתוך המספר המחושב 2087489. באופן דומה היינו יכולים לשלוף ספרה מהסוף ושתי ספרות מההתחלה, או כל צירוף אחר שבעזרתו נוכל למנוע התנגשויות.

פונקציית שינוי בסיס: נוכל לקחת מפתח נתון ולהמיר אותו לבסיס אחר. לדוגמה, נחליט כי את המספר הנתון (בבסיס 10, כמובן) נמיר למספר בבסיס 8. בדרך זו, מפתח שערכו 137 בבסיס עשרוני יהיה 211 בבסיס שמונה.

פונקציית קיפול: פונקציה זו "מקפלת" מספר (מפתח) נתון לשניים (או יותר). כיצד עושים זאת? נאמר כי המפתח הוא 149625. נחלק את המספר הזה לשני חלקים: 625 - החלק התחתון של המספר ו-149 - חלקו העליון. עתה נחשב את סכום שני החלקים:

```
  625
= 149
  774
```

בצורה זו קיבלנו ערך חדש, 774.

נבחן דוגמה נוספת: נתון המספר 3917238123. נחלק אותו לארבעה חלקים: 123, 238, 917 ו-003. נחשב את הסכום הכולל:

```
  123
 +238
  917
    3
 1281
```

המפתח שקיבלנו הוא 1281. אם היו מגדירים בפנינו כי תחום המפתח הוא 0...999 היה עלינו לחזור שוב על תהליך הקיפול, כדי לקבל תוצאה קטנה מ-999. במקרה החדש היינו מקבלים את המפתח 282.

שלוש הפונקציות האחרונות שהצגנו מדגימות טכניקות שונות למניעת התנגשות. חוקרים בתחומי המתמטיקה ומדעי המחשב מנסים למצוא פתרונות שונים לבעיה. לדוגמה, קיימת שיטות הנקראות CRC ו-ECCM שבעזרתן ניתן להקטין את הסיכוי להתנגשות עד לרמה סבירה. למרות זאת, ביצוע תהליכים כאלה דורשים זמן עיבוד רב. לכן, כאשר אנו בוחרים בפונקציית ערבול כלשהי, עלינו להחליט אם היא מתאימה, ואם כדאי להשקיע מאמצים בפיתוחה. יש לדעת, כי מספר ההתנגשויות אינו קשור דווקא לפונקציית הערבול, אלא לאופי מספרי הזיהוי (מפתחות), הפיזור שלהם על פני טווח המספרים, צפיפות ועוד. נושאים אלה חורגים מתחום הדיון שלנו, ואנו משאירים זאת כאתגר לקורא המתעניין.

נספח ב

פתרונות לתרגילים

פרק 2

תרגיל 2.1

```
program Targil1;
var
    A, B, C:     Integer;
    Total:       Real;

begin
    Writeln ('Enter three numbers (A, B, C):');
    Readln (A, B, C);
    Total := (A + B + C) / 3;
    Writeln ('Result = ', Total);
end.
```

תרגיל 2.2

```
program Targil2;
var
    A, B, C:       Integer;

begin
    Writeln ('Enter three numbers (A, B, C): ');
    Readln (A, B, C);
    Writeln ('Result = ', A mod (B + C));
end.
```

תרגיל 2.3

```
program Targil3;
var
    W:              Real;

begin
    Writeln ('Enter weight in tons:');
    W := W * 1000000;
    Writeln ('Weight in grahms: ', W);
end.
```

תרגיל 2.4

```
program Targil4;
var
    A, B, C:       Real;

begin
    Writeln ('Type in three real values (A, B, C):');
    Readln (A, B, C);
    Writeln ('Sum = ', (A + B + C));
    Writeln ('Multiplaction = ', (A * B * C));
    Writeln ('Average = ', ((A + B + C) / 3));
end.
```

תרגיל 2.5

```
program Targil5;
var
    R:              Real;

begin
    Writeln ('Enter a number: ');
    Readln (R);
    R := R * (-1);
    Writeln ('The reversed value: ', R);
end.
```

תרגיל 2.6

```
program Targil6;
var
   I:          Integer;

begin
   Writeln ('Type in a value:');
   Readln (I);
   Writeln ('Low digit: ', I mod 10);
   Writeln ('High digit: ', I div 10);
end.
```

תרגיל 2.7

```
program Targil7;
var
   R:          Real;

begin
   Writeln ('Type in real value:');
   Readln (R);
   R := Abs (R);
   Writeln ('Square = ', Sqr (R));
   Writeln ('Square root = ', Sqrt (R));
end.
```

תרגיל 2.8

```
program Targil8;
var
   Ch:          Char;

begin
   Writeln ('Type in a single char:');
   Readln (Ch);
   Writeln ('The ASCII value of ', Ch, ' is ', Ord (Ch));
end.
```

תרגיל 2.9

```
program Targil9;
var
    Ch:         Char;

begin
    Writeln ('Type in a single char:');
    Readln (Ch);
    Ch := Chr (Ord (Ch) + 1);
    Writeln ('Next char in ASCII table is ', Ch);
end.
```

תרגיל 2.10

```
program Targil10;
var
    R:          Real;

begin
    Writeln ('Type in a real value:');
    Readln (R);
    Writeln ('The fraction of ', R, ' is ', Frac (R));
end.
```

תרגיל 2.11

```
program Targil11;
var
    H, W, L, Size: Real;

begin
    Writeln ('Enter three dimensions of a cube (H, W, L):');
    Readln (H, W, L);
    Size := H * W * L;
    Writeln ('The volume of the cube is ', Size);
end.
```

תרגיל 2.12

```
program Targil12;
var
   R:          Real;

begin
   Writeln ('Type in a real value:');
   Readln (R);
   R := R - Int (R);
   Writeln ('The fraction = ', R);
end.
```

תרגיל 2.13

```
program Targil13;
var
   I:          Integer;

begin
   Writeln ('Type in an integer value:');
   Readln (I);
   Writeln ('Result: ', I div Abs (I));
end.
```

תרגיל 2.14

```
program Targil14;
var
   Number:     Integer;
   Groups:     Integer;
   LeftOver:   Integer;

begin
   Number := 37;
   Groups := Number div 5;
   LeftOver := Number mod 5;
   Writeln ('Number of groups: ', Groups);
   Writeln ('Students without a group: ', LeftOver);
end.
```

פרק 3

תרגיל 3.1

```
program Targil1;
var
   A, B, C:      Integer;
   Result:       Boolean;

begin
   Writeln ('Type in three integer values:');
   Readln (A, B, C);
   Result := (A  = B) or (A = C) or (B = C);
   Writeln ('Result = ', Result);
end.
```

תרגיל 3.2

```
program Targil2;
var
   A, B, C:      Integer;
   Result:       Boolean;

begin
   Writeln ('Type in three integer values:');
   Readln (A, B, C);
   Result := (A < B) and (B < C);
   Writeln ('Result = ', Result);
end.
```

תרגיל 3.3

```
program Targil3;
var
   A, B:         Integer;
   Result:       Boolean;

begin
   Writeln ('Type in two integer values:');
   Readln (A, B);
   Result := (A = 0) and (B = 0);
   Writeln ('Result = ', Result);
end.
```

פרק 4

תרגיל 4.1

```
program  Targil1;
var
   W, H:         Integer;
   A, B:         Integer;

begin
   Writeln ('Enter  width, height (W, H):');
   Readln (W, H);
   for  A := 1  to  H  do
      begin
         for  B := 1  to  W  do
            Write ('*');
         Writeln;
      end;
end.
```

תרגיל 4.2

```
program  Targil2;
var
   R:          Real;
   Largest:  Real;
   I:          Integer;
   Ch:         Char;

begin
   repeat
      Largest := 0;
      for  I := 1  to  10  do
        begin
         Writeln ('Enter next value: ');
         Readln (R);
         if  Frac (R)  >  Largest  then
            Largest := Frac (R);
        end;
      Writeln ('Largest fraction = ', Largest);
      Writeln ('Do you want to repeat again?');
      Readln (Ch);
   until  Ch  =  'N';
end.
```

תרגיל 4.3

```
program  Targil3;
var
   A, B:         Integer;

begin
   repeat
      Writeln ('Enter value:');
      Readln (A);
   until  (A  >=  1)  and  (A  <=  100);
   for  B := 1  to  25  do
      Writeln ('');
   repeat
      Writeln ('Player #2, enter your guess:');
      Readln (B);
      if  B  >  A  then
            Writeln ('Too high.');
      if  B  <  A  then
            Writeln (Too low.');
   until  A  =  B;
   Writeln ('Very good !!!');
end.
```

תרגיל 4.4

```
program Targil4;
var
   A, B:         Integer;

begin
   Writeln ('Type in two integer values:');
   Readln (A, B);
   if A = B then
      Writeln ('Equal !!!')
   else
      Writeln ('Not equal !!!');
end.
```

תרגיל 4.5

```
program Targil5;
var
   I:          Integer;

begin
   Writeln ('Type in an integer value:');
   Readln (I);
   if I = 0 then  Writeln ('ZERO');
   if I = 1 then  Writeln ('ONE');
   if I = 2 then  Writeln ('TWO');
   if I = 3 then  Writeln ('THREE');
   if I = 4 then  Writeln ('FOUR');
end.
```

תרגיל 4.6

```
program Targil6;
var
   A, B, C:     Integer;

begin
   Writeln ('Type three integer values:');
   Readln (A, B, C);
   if  (A  >=   B)  and  (A  <=  C)  then
      Writeln ('OK')
   else
      Writeln ('NOT OK');
end.
```

תרגיל 4.7

```
program Targil7;
var
   L, H, Counter:     Integer;

begin
   Writeln ('Type in the low and high values:');
   Readln (L, H);
   if  L < H  then
      for  Counter := L  to  H  do  Writeln (Counter)
   else
      for  Counter := L  downto  H  do  Writeln (Counter);
end.
```

תרגיל 4.8

```
program Targil8;
var
   L, H, Counter:     Integer;

begin
   Writeln ('Type the low and high values:');
   Readln (L, H);
   for  Counter := L  to  H  do  Writeln (Counter);
   { L  >  H   שורה זו לא תתבצע אם}
   for  Counter := L  downto  H  do  Writeln (Counter);
   { L  <  H   שורה זו לא תתבצע אם}
end.
```

תרגיל 4.9

```
program Targil9;
var
   A, B:         Integer;

begin
   Writeln ('Type in two integer values (low, high):');
   Readln (A, B);
   repeat
      Writeln (A);
      A := A  +  1;
   until  A = B;
end.
```

תרגיל 4.10

```
program Targil10;
var
   Counter, Total:   Integer;
   R:                Real;

begin
   for Counter := 1 to 10 do
      begin
            total := 0;
            Writeln ('Type in a real value:');
            Readln (R);
            if Frac (R) <> 0 then Total := Total + 1;
      end;
   Writeln ('Number of values counted: ', Total);
end.
```

תרגיל 4.11

```
program Targil11;
var
   I:          Integer;

begin
   Writeln ('Type in an integer value:');
   Readln (I);
   if  I < 0  then  I := I * (-1);
   Writeln ('Absolute value: ',  I);
end.
```

תרגיל 4.12

```
program Targil12;
var
   A, B:       Integer;

begin
   Writeln ('Type in two values:');
   Readln (A, B);
   if  A > B  then
      Writeln ('The largest value is: ', A)
   if B > A then
      Writeln ('The largest value is: ', B);
end.
```

תרגיל 4.13

```
program Targil13;
var
   I:          Integer;

begin
   Writeln ('Type in an integer value:');
   Readln (I);
   if  I mod 2 = 0  then
      Writeln ('Even value !!!')
   else
      Writeln ('Odd value !!!');
end.
```

תרגיל 4.14

```
program Targil14;
var
    Ch:            Char;

begin
    Writeln ('Type in a single char:');
    Readln (Ch);
    if ((Ch >= 'A') and (Ch <= 'Z')) or
        ((Ch >= 'a') and (Ch <= 'z')) then
        Writeln ('Char !!!');
end.
```

תרגיל 4.15

```
program Targil15;
var
    A, B, C:       Real;
    Delta:     Real;

begin
    Writeln ('Type in 3 real numbers for the equation:');
    Readln (A, B, C);
    Delta := Sqr (B) - 4 * A * C;
    if  Delta >= 0 then
        begin
                R1 := -B + Sqrt (Delta);
                R2 := -B - Sqrt (Delta);
                Writeln ('root 1 = ', R1);
                Writeln ('root 2 = ', R2);
        end
    else
        Writeln ('Cannot calculate roots.');
end.
```

תרגיל 4.16

```
program Targil16;
var
    A, B, C:       Real;

begin
    Writeln ('Type in the values for the Nitzavim:');
    Readln (A, B);
    C := Sqrt (Sqr (A) + Sqr (B));
    Writeln ('Yeter = ', C);
end.
```

תרגיל 4.17

```
program Targil17;
var
    A:            Integer;

begin
    Write ('The angle is: ');
    Readln (A);
    if (A > 0) and (A < 180) then
        begin
                if A < 90 then Writeln ('זווית חדה');
                if A = 90 then Writeln ('זווית ישרה');
                if A > 90 then Writeln ('זווית קהה');
        end
    else
        Writeln ('זווית שגויה.');
end.
```

תרגיל 4.18

```
program Targil18;
var
    I:            Integer;

begin
    Writeln ('Type an integer value:');
    Readln (I);
    repeat
        Write (I mod 10);
        I := I div 10;
    until  I = 0;
end.
```

תרגיל 4.19

```
program Targil19;
var
    Ch:           Char;

begin
    Writeln ('Type in a character:');
    Readln (Ch);
    case Ch of
        '0'..'9': Writeln ('Digit');
        'A'..'Z': Writeln ('Capital letter');
```

```
        'a'..'z': Writeln ('Small letter');
        else      Writeln ('Invalid');
    end;
end.
```

פרק 5

תרגיל 5.1

```
prograem Targil1;
var
    A, B, C, Temp: Integer;

begin
    Writeln ('Enter three values:');
    Readln (A, B, C);
    if  A > B  then
        begin
                Temp := A;
                A := B;
                B := Temp;
        end;

    if  B > C  then
        begin
                Temp := B;
                B := C;
                C := Temp;
        end;

    if  A > B  then
        begin
                Temp := A;
                A := B;
                B := Temp;
        end;

    Writeln ('Highest value: ', C);
    Writeln ('Average value: ', B);
    Writeln ('Lowest value: ', A);
end.
```

תרגיל 5.2

```
program Targil2;
var
   N:          Integer;

begin
   Readln (N);
   repeat
      Writeln (N mod 10);
      N := N div 10;
   until  N = 0;
end.
```

תרגיל 5.3

```
program Targil3;
var
   A, B, C:      Integer;
   R:         Real;
   Counter: Integer;

begin
   A := 0;
   B := 0;
   C := 0;

   for  Counter := 1  to  100  do
      begin
            Writeln ('Enter new value: ');
            Readln (R);
            if  Int (R) <> 0  then  A := A + 1;
            if  Frac (R) <> 0  then  B := B + 1;
            if  R = 0  then  C := C + 1;
      end;

   Writeln ('Values included integer value: ', A);
   Writeln ('Values included fraction: ', B);
   Writeln ('Values equal to zero: ', C);
end.
```

תרגיל 5.4

```
program Targil4;
var
   R:             Real;
   Counter: Integer;

begin
   Total := 0.0;

   for  Counter := 1  to  10  do
      begin
            Writeln ('Enter new value:');
            Readln (R);
            Total := Total + Frac (R);
      end;

   Writeln ('Total value: ', Total);
end.
```

תרגיל 5.5

```
program Targil5;
var
   A, B, Total:   Integer;
   Counter:       Integer;

begin
   Writeln ('Enter two values: ');
   Readln (A, B);
   Total := 0;
   for  Counter := 1  to  A  do
      Total := Total + B;
   Writeln ('Total value', total);
end.
```

תרגיל 5.6

```
program Targil6;
var
   R, Total:      Real;

begin
   Total := 0.0;
```

```
    repeat
        Writeln ('Enter next real value:');
        Readln (R);
        Total := Total + R;
    until  R = 0.0;

    Writeln ('Total value ', total);
end.
```

תרגיל 5.7

```
program Targil7;
var
    Ch:           Char;
    Total:        Integer;

begin
    Total := 0;

    repeat
        Readln (Ch);
        if  Ch = 'A'  then  Total := Total + 1;
    until  Ch = '.';

    Writeln ('Counter = ', Total);
end.
```

תרגיל 5.8

את הכנת תרשים הזרימה אנו משאירים לך כאתגר אישי.

פרק 6

תרגיל 6.1

```
program Targil1;
const
    Total:      Real = 0;
var
    I:          Integer;

procedure AddValue (Value : Integer);
begin
    Total := Total + Value ;
end;
```

```
begin
   for I := 1 to 10 do
      begin
            Writeln ('Enter mark of student number ', I);
            Readln (N);
            AddValue (N);
      end;

   Writeln ('Average = ', Total / 10);
end.
```

תרגיל 6.2

```
program Targil2;
var
   X, Y:          Integer;

procedure FindBigger (A, B : Integer);
begin
   if  A > B  then
      Writeln ('Bigger: ', A)
   else
      Writeln ('Bigger: ', B);
end;

begin
   Writeln ('Enter two values:');
   Readln (X, Y);
   FindBigger (X, Y);
end.
```

תרגיל 6.3

```
program Targil3;
const
   LastValue:     Integer = 0;
var
   NewValue:      Integer;

procedure Check (N : Integer);
begin
   if  N > 0  then
      begin
            if  N < LastValue  then
               Writeln ('Error, invalid value !')
```

```
            else
                LastValue := N;
        end;
end;

begin
    repeat
        Writeln ('Enter new value:');
        Readln (NewValue);
        Check (NewValue);
    until  NewValue < 0;
end.
```

תרגיל 6.4

```
program Targil4;
var
    A, B, C:        Integer;
    Largest:       Integer;

procedure Kelet;
begin
    Writeln ('Enter three values (A, B, C):');
    Readln (A, B, C);
end;

function Big (X, Y : Integer) : integer ;
begin
    if  X > Y  then
        Big := X
    else
        Big := Y;
end;

procedure FindBig;
var
    Largest:       Integer;

begin
    { מציאת הערך הגדול מבין  שני הערכים הראשונים }
    Largest := Big (A, B);
    { מציאת הערך הגדול מבין שני אלה שנבחנו עתה וערך נוסף }
    Largest := Big (Largest, C);
    Writeln (Largest);
end;
```

```
begin
    Kelet;
    FindBig;
end.
```

תרגיל 6.5

```
program Targil5;
var
    A, B, students:         Integer;

function ClassNumber : Integer;
var
    N:          Integer;

begin
    Writeln ('Enter the number of classes in school:');
    Readln (N);
    ClassNumber := N;
end;

function StudentsNumber : Integer;
var
    N:          Integer;

begin
    Writeln ('Enter the number of students:');
    Readln (N);
    StudentsNumber := N;
end;

procedure Chishov (Number : Integer);
var
    I, N:       Integer;
    Total:      Real;

begin
    Total := 0.0;
    for  I := 1  to  Number  do
        begin
            Writeln ('Enter new mark:');
            Readln (N);
            Total := Total + (N / Number);
        end;
    Writeln ('Average of class: ', Total);
end;
```

```
begin
   for  A := 1  to  ClassNumber  do
      begin
            Students := StundetsNumber;
            Chishov (Students);
      end;
end.
```

תרגיל 6.6

```
program Targil6;
var
   X, Y, Z:       Real;

function Delta (A, B, C : Real) : Real;
var
   Temp:          Real;

begin
   Temp := Sqr (B) - 4*A*C;
   if  Temp >=0  then
      Delta := Sqrt (Temp)
   else
      Delta := -1;
end;

begin
   Writeln ('Enter three constants of square equasssion:');
   Readln (X, Y, Z);
   Writeln ('Delta = ', Delta (X, Y, Z));
end.
```

תרגיל 6.7

```
program Targil7;
var
   N:             Integer;

function CalcTotal (Value : Integer) : Integer;
var
   Temp:          Integer;

begin
   Temp := 0;
   repeat
      Temp := Temp  +  (Value mod 10);
```

```
      Value := Value div 10;
   until  Value = 0;
   CalcValue := Temp;
end;

begin
   Writeln ('Enter value:');
   Readln (N);
   Writeln ('Sum of all digits: ', CalcTotal (N));
end.
```

תרגיל 6.8

```
program Targil8;
function Pi : Real;
var
   N:              Integer;
   Item, Total:   Real;

begin
   N:=1;
   Total := 0;
   repeat
      Item := 1 / ((N - 1) * 4 + 1) - 1 / (N * 4 - 1);
      Total := Total + Item;
      n := n + 1;
   until  Item < 0.0001;
   Total := 4 * Total;
   Pi := Total;
end;

begin
   Writeln ('Pi = ', Pi);
end.
```

תרגיל 6.9

```
program Targil9;
var
   Payment, Tax:    Real;
   Left:                 Real;

procedure Calc (P, T : Real; var L : Real);
begin
   L := P * (1 - T / 100);
end;
```

```
begin
    Writeln ('Enter workers payment and tax to pay:');
    Readln (Payment, Tax);
    Calc (Payment, Tax, Left);
    Writeln ('Total left after reduction: ', Left);
end.
```

פרק 7

תרגיל 7.1

```
program Targil1;
var
    Name:         string;

begin
    Writeln ('Type in your name: ');
    Readln (Name);
    Writeln ('Have a nice day, ', Name);
end.
```

תרגיל 7.2

```
program Targil2;
var
    St1, St2:     string;

begin
    St1 := 'HELLO';
    Readln (St2);
    if  St1 = St2  then
        Writeln ('Hello to you too !!!');
end.
```

תרגיל 7.3

```
program Targil3;
var
    St:           string;
    P:            Integer;

begin
    Writeln ('Type in a string:');
    Readln (St);
    Writeln ('Type the position for the string');
```

```
    Readln (P);
    Writeln  ('St[', P, '] = ', St[P]);
end.
```

תרגיל 7.4

```
program targil4;
var
    St:          string;
    I:           Integer;

procedure Print;
begin
    for  I := 1  to  Length(St)  do
        Write (St[I], '  ');
end;

begin
    Writeln ('Type a string:');
    Readln (st);
    Print;
end.
```

תרגיל 7.5

```
program Targil5;
var
    St:          string;
    P:           Integer;

begin
    Writeln  ('Type a string:');
    Readln (St);
    Writeln ('Enter the position within the string limits:');
    Readln (P);
    if  P > Length (st)  then
        Writeln ('Invalid value -- ', p)
    else
        Writeln ('St[', P, '] = ', St[P]);
end.
```

תרגיל 7.6

```
program Targil6;
var
    St1, St2:      string;

begin
    Writeln ('Type a string:');
    Readln (St1);
    if  Length (St1) >= 3  then
        begin
                St2 := Copy (St1, Length(St) - 1, 2);
                Writeln ('Last two characters of the string: ', St2);
        end
    else
        Writeln ('Error: string is too short !!!');
end.
```

תרגיל 7.7

```
program Targil7;
var
    St1, St2:      string;
    P:           Integer;

procedure GetString;
begin
    Writeln ('Type a string:');
    Readln (St1);
end;

procedure WriteWord;
begin
    P := Pos ('  ', St1);
    if  P = 0  then
        St2 := St1
    else
        St2 := Copy (St1, 1, P);
        Writeln ('First (or only word): ', st2);
end;

begin
    GetString;
    WriteWord;
end.
```

תרגיל 7.8

```
program Targil8;
var
   St:        string;
   Result:    Integer;
   Error:     Integer;

begin
   Writeln ('Enter a value:');
   Readln (St);
   Val (St, Result, Error);
   if  Error = 0  then
      Writeln ('Everything is ok !!!')
   else
      Writeln ('Error: ', St[Error]);
end.
```

תרגיל 7.9

```
program Targil9;
var
   I:         Integer;
   St:        string;

function Translate (A : Integer) : string;
var
   S:         string;

begin
   Str (A, S);
   A := Length(S) - 2;
   while  A > 1  do
      begin
            Insert (', ', S, A);
            A := A - 3;
      end;
   Translate := S;
end;

begin
   Writeln ('Type an integer value:');
   Readln (I);
   Writeln ('Result: ', Translate(St));
end.
```

תרגיל 7.10

```
program Targil10_1;
var
   I:            Integer;

function TotalValue (A : Integer) : Integer;
var
   St1, St2: string;
   Total:     Integer;
   Error:     Integer;
   Value:        Integer;

begin
   Str (A, St1);
   Total := 0;
   for  A := 1  to  Length (St1)  do
      begin
            St2 := St1[I];
            Val (St2, Value, Error);
            Total := Total  +  Value;
      end;
end;

begin
   Writeln ('Enter an integer value (lager than 0): ');
   Readln (I);
   Writeln ('Sum of  all digits: ', TotalValue(I));
end.

program Targil10_2;
var
   I:          Integer;
   St:         string;

begin
   Writeln ('Enter an integer value:');
   Readln (I);
   Str (I, St);
   for  I := Length (St)  downto  1  do
      Write (St[I]);
end.
```

תרגיל 7.11

```
program Targil11;
var
   st:            string;

function SetString (S : string) : string;
var
   I, N:          Integer;

begin
   I := 1;
   while  I < Length (S) - 1  do
      begin
            while  S[I] = S[I + 1]  do
               Delete (s, I, 1);
               I := I + 1;
      end;
   SetString := S;
end;

begin
   Writeln ('Enter a string:');
   Readln (st);
   Writeln ('String: ', SetString(st));
end.
```

תרגיל 7.12

```
program Targil12;
var
   St1, St2:     string;
   I:            Integer;
   Len:          Integer;
   Counter:      Integer;
   Longest:      Integer;
   Shortest:     Integer;
   Avarage:      Integer;

begin
   Writeln ('Type a string:');
   Readln (St1);
   Len := Length (St1);
```

```
    Counter := 0;
    Shortest := 255;
    Longest := 0;
    while  Length (St) > 0  do
       begin
             I := Pos (' ', St1);
             St2 := Copy (St1, 1, I);
             Delete (St1, 1, I);
             Counter := Counter  +  1;

             if Longest < Length (St2) then
                 Longest:=Length (St2);

             if Shortest < Length (St2) then
                 Shortest:=Length  (St1);
       end;

    { אורך מילה ממוצע  =  (אורך כולל  -  מספר רווחים)  /  מספר המילים במשפט }
    Avarage := Len - (Counter - 1) / Counter;

    Writeln ('String length: ',  Len);
    Writeln ('Number of  word(s): ', Counter);
    Writeln ('Shortest word: ', Shortest);
    Writeln ('Longest word: ', Longest');
    Writeln ('Avarage length of word: ', Avarage);
end.
```

תרגיל 7.13

```
program Targil13;
var
    St:          string;
    I:           Integer;

begin
    Writeln ('Type a string:');
    Readln  (St);
    repeat
       I := Pos ('[רווח כפול]', St);
       if  I > 0  then
             Delete (st, I, 1);
    until  I = 0;

    if  St[1] = ' '  then
       Delete (St, 1, 1);
```

```
    if  St[Length(st)] = ' '  then
        Delete (St, Length(St), 1);

    Writeln ('New string: ', St);
end.
```

פרק 8

תרגיל 8.1

```
program Targil1;
var
    Values:     array [1..10]  of   Integer;
    I, N:       Integer;

begin
    for  I := 1  to  10  do
        Values[I] := 0;
    Writeln ('Enter position with in the limits (1-10): ');
    Readln (I);
    Writeln  ('Enter value to put in cell: ');
    Readln (N);
    if  (I >= 1)  and  (I <= 10)  then
        Values[I] := N
    else
        Writeln ('Error: Position out of  array limits.');
end.
```

תרגיל 8.2

```
program Targil2;
var
    Values:       array [1..5]  of  Real;
    I:            Integer;
    R:            Real;

begin
    for  I := 1  to  5  do
        begin
                Writeln ('Enter value for cell #', I);
                Readln (Values[I]);
        end;
```

```
   for  I := 1  to  5  do
      begin
            R := Values[I] / 100;
            Writeln ('Value #', I, ' after divission: ', R);
      end;
end.
```

תרגיל 8.3

```
program Targil3;
var
   Num:         array [1..10]  of  Integer;
   Higher:      Integer;
   Lower:       Integer;
   Equal:       Integer;
   I:           Integer;

begin
   Higher := 0;
   Lower := 0;
   Equal := 0;

   for  I := 1  to  10  do
      begin
            if  Num[I]  >  0  then
               Higher := Higher  +  1;
            if  Num[I]  =  0  then
               Equal := Equal  +  1;
            if  Num[I]  <  0  then
               Lower := Lower  +  1;
      end;

   Writeln  ('Total values higher than zero: ', Higher);
   Writeln  ('Total values equal to zero: ', Equal);
   Writeln  ('Total values lower than zero: ', Lower);
end.
```

תרגיל 8.4

```
program Targil4;
var
   Numbers:            array [1..20]  of  Integer;
   I, Value, Counter:  Integer;
```

```
begin
   Writeln ('Type in a value:');
   Readln (V);

   Counter := 0;
   for  I := 1  to  20  do
      begin
            if  Numbers[I] = Value  then
                Counter := Counter + 1;
      end;

   if  Counter > 0  then
      Writeln ('Value appear ', Counter, ' time(s) in the array.')
   else
      Writeln ('Value does not appear at all !');
end.
```

תרגיל 8.5

```
program Targil5;
var
   Values:               array [0..9]  of  Integer;
   I, Total1, Total2:    Integer;

begin
   for  I := 0  to  9  do
      begin
            Writeln ('Enter value for #', I, ' cell.');
            Readln (Values[I]);
      end;

   Total1 := 0;
   Total2 := 0;
   for  I := 0  to  4  do
      begin
            Total1 := Total1 + Values[I * 2];
            Total2 := Total2 + Values[I * 2 + 1];
      end;

   if  Total1 = Total2  then
      Writeln ('Equal !!!')
   else
      Writeln ('Not equal.');
end.
```

תרגיל 8.6

```
program Targil6;
var
    Numbers:    array [0..9]  of  Integer;
    I,  Value:     Integer;

begin
    Writeln ('Enter a starting value:');
    Readln (Value);
    for  I := 0  to  9  do
        Numbers[I] := I + Value;
end.
```

תרגיל 8.7

```
program Targil7;
var
    Numbers:   array [1..10]  of  Integer;
    Total, I:      Integer;

begin
    for  I := 1  to  10  do
        begin
                Writeln ('Enter value for cell #', I);
                Readln (Numbers[I]);
        end;

    for  I :=  1  to  10  do
        Total := Total  +  Numbers[I];
    Writeln ('Sum of all values: ', Total);
end.
```

תרגיל 8.8

```
program Targil8;
var
    Values:   array [1..10]  of  Integer;
    I:            Integer;

begin
    for  I := 1  to  10  do
        begin
                Writeln ('Enter value to cell #', I);
                Readln (Values[I]);
        end;
```

```
    Writeln  ('Values from the buttom to the top:');
    for  I := 10  downto  1  do
        Writeln (Values[I]);
end.
```

תרגיל 8.9

```
program Targil9;
var
    Values:   array [1..10]  of  Integer;
    I, N:       Integer;
    OK:             Boolean;

begin
    OK := True;
    N := Values[1];
    for  I := 2  to  10  do
        if  Values[I] <> N  then  OK := False;
        if  OK = True  then
                Writeln ('All values at the array are equal !')
        else
                Writeln ('Not all values at the array are equal.');
end.
```

תרגיל 8.10

```
program Targil10;
var
    Numbers:    array [1..10]  of  Integer;
    I, N:           Integer;
    Exist, Total:             Boolean;

begin
    for  I := 1  to  10  do
        begin
                Writeln ('Type a value for cell no. #',  I);
                Readln (Numbers[I]);
        end;

    for  N := 1  to  10  do
        begin
                Exist := False;
                for  I := 1  to  10  do
                    if  Numbers[I] = N  then  Exist := True;
                    if  Exist = False  then
                            Total := False;
        end;
```

```
   if  Total = True  then
      Writeln ('All values from 1 to 10 exist !')
   else
      Writeln ('Not all values from 1 to 10 exist.');
end.
```

תרגיל 8.11

```
program Targil11;
var
   Numbers:        array [1..10]  of  Integer;
   I:              Integer;
   First, Second:  Integer;

begin
   for  I := 1  to  10  do
      begin
            Writeln ('Type an integer value for cell no. ',I);
            Readln (Numbers[I]);
      end;

   First := Numbers[1];
   Second := First;
   for  I := 2  to  10  do
      if  Numbers[I] > First  then
            begin
               Second := First;
               First := Numbers[I]
            end
      else
            if  Numbers[I]  >  Second  then
               Second  :=  Numbers[I];
   Writeln ('Second lowest value is: ', Second);
end.
```

תרגיל 8.12

```
program Targil12;
var
   Numbers:             array [1..10]  of  Integer;
   I,  N,  Value:       Integer;
   EndLoop:    Boolean;

begin
   for  I := 1  to  10  do
```

```
        begin
            Writeln ("Enter an integer value for cell no. ', I);
            Readln (Numbers[I]);
        end;

    N := 1;
    EndLoop := False;
    repeat
        I := 1;
        repeat
            if  (I <> N) and (Numbers[I] = Numbers[N])          then
                begin
                        Value := Numbers[I];
                        EndLoop := True;
                end;
            I := I  +  1;
        until  (I = 10)  or  (EndLoop = True);
        N := N  +  1;
    until  (N = 10)  or  (EndLoop = True);

    if  EndLoop = True  then
        Writeln ('First value repeating more than once: ', Value)
    else
        Writeln ('No value is repeating on itself.');
end.
```

תרגיל 8.13

```
program Targil13;
var
    Num1, Num2:    array [1..10]  of  Real;
    I:                     Integer;

begin
    for  I := 1  to  10  do
        Num1[I] := Num1[I] + Num2[I];

    for  I := 1  to  10  do
        Writeln ('Sum of cells no. ', I, ' is: ', Num1[I]:0:2);
end.
```

פרק 9

תרגיל 9.1

```
program PrintTable;
var
   A, B, X:      Integer;
   Table:        array [1..4, 1..4]  of  Boolean;

begin
   for  A := 1  to  4  do
      for  B := 1  to  4  do
            begin
               Write (A, '--', B, ' = ');
               Readln (X);
               if  X = 0  then
                        Table[A, B] := False
               else
                        Table[A, B] := True;
            end;

   for  A := 1  to  4  do
      begin
            for  B := 1  to  4  do
               if  Table[A, B] = True  then
                        Write ('1')
               else
                        Write ('0');
            Writeln;
      end;
end.
```

תרגיל 9.2

```
program SumTable;
var
   A, B, Sum:     Integer;
   Table:         array [1..3, 1..4]  of  Integer;

begin
   for  A := 1  to  4  do
      for  B := 1  to  4  do
            begin
               Write (A, '--', B, ' = ');
               Readln (Table[A, B]);
            end;
```

```
        for  A := 1  to  4  do
                begin
                    for  B := 1  to  4  do
                              Sum := Sum + Table[A, B];
                              Writeln ('Sum of row no. ', A, ' = ', Sum);
                end;

        for  A := 1  to  4  do
                begin
                    for  B := 1  to  4  do
                              Sum := Sum + Table[B, A];
                              Writeln ('Sum of column no. ', A, ' = ', Sum);
                end;
end.
```

תרגיל 9.3

```
program Matrix;
var
    A, B:         Integer;
    Matrix:       array [1..5, 1..5]  of  Real;
    Table:        array [1..25]  of  Real;

begin
    for  A := 1  to  4  do
        for  B := 1  to  4  do
                Table[A * B] := Matrix[A, B];
end.
```

תרגיל 9.4

```
program ReadString;
var
    Ma:           array [1..255]  of  integer;
    St:        String;
    A, B, L:   Integer;
    Tav:       Char;

begin
    for A := 1 to 255 do Ma[A] := 0;
    Writeln ('Insert a string:');
    Readln (St);
    L := length (St);
    for A := 1 to L do
        B := ord (St[A]);
        Ma := Ma[B] + 1;
end;
```

```
   for A := 1  to  255 do
      begin
            if Ma[A] > 0 then
               begin
                     Tav := chr (A);
                     Writeln (Tav, ' - ', Ma[A]);
               end;
      end.
   Readln;
```

תרגיל 9.5

```
program CopyString;
var
   A, B:        Integer;
   St:          String;
   StrTable:    array [1..4]  of  string;
   Table:       array [1..4, 1..5]  of  Char;

begin
   for  A := 1  to  4  do
      begin
            Writeln ('Type string no. ', A);
            Readln (St);
            StrTable[A] := Copy (St, 1, 5);
      end;
   for  A := 1  to  4  do
      for  B := 1  to  4  do
            Table[A, B] := StrTable[A][B];
end.
```

תרגיל 9.6

```
program TableScan;
var
   Found:              Boolean;
   A, B, Counter:      Integer;
   Table1, Table2:     array [1..40]  of  Integer;

begin
   Counter := 0;
   for  A := 1  to  40  do
      begin
            Found := False;
            for  B := 1  to  Counter do
               if  Table1[A] = Table2[B]  then  Found := True;
```

```
                if  Found = False  then
                        begin
                      Counter := Counter  +  1;
                        Table2[Counter] := Table1[A];
                        end;
      end;
end.
```

תרגיל 9.7

```
program PrintStrings;
var
   I:              Integer;
   Start:          Boolean;
   StrTable:               array [1..10]  of  String;

begin
   Start := False;
   for  I := 1  to  10  do
      begin
            if  StrTable[I] = 'START'  then
               Start := True
            else
               if  StrTable[I] = 'STOP'  then
                       Start := False
               else
                       if  Start = True  then
                       Writeln (StrTable[I]);
      end;
end.
```

תרגיל 9.8

```
program TableScan;
var
   A, B, TempLen:       Integer;
   HighestLen:          Integer;
   Table:               array [1..10, 1..10]  of  Integer;

begin
   HighestLen := 0;

   { Part one - search in all rows }
   for  A := 1  to  10  do
      begin
```

```
            for  B := 1  to  9  do
                begin
                        C := B;
                        TempLen := 1;
                        while  (C <= 10) and (Table[A, B] = Table[A, C])  do
                      begin
                         C := C + 1;
                        TempLen := TempLen  + 1;
                        end;
                   if  HighestLen < TempLen  then
                      begin
                         X := B;
                         Y := A;
                         HighestLen := TempLen;
                      end;
                end;
      end;

   { Part two - search in all columns }
   for  A := 1  to  10  do
      begin
            for  B := 1  to  9  do
               begin
                   C := B;
                   TempLen := 1;
                   while  (C <= 10) and (Table[B, A] = Table[C, A])  do
                      begin
                         C := C + 1;
                         TempLen := TempLen  + 1;
                      end;

                   if  HighestLen < TempLen  then
                      begin
                         X := A;
                         Y := B;
                         HighestLen := TempLen;
                      end;
               end;
      end;

   { Output data }
   Writeln ('Longest sequence found: ', HighestLen);
   Writeln ('Found at position: ', A, ', ', B);
   Writeln ('Value is: ', Table[A, B]);
end.
```

פרק 10

תרגיל 10.1

```
program SearchOK;
var
   I:           Integer;
   Found:   Boolean;
   Table:    array [1..10]  of  string;

begin
   for  I := 1  to  10  do
      begin
            Writeln ('Enter string number ', I);
            Readln (Table[I]);
      end;

   Found := False;
      for  I := 1 to  10  do
            if  Table[I] = 'OK'  then
               Found := True;

            if  Found = True  then
               Writeln  ('OK !!!')
            else
               Writeln  ('NOT OK.');
end.
```

תרגיל 10.2

```
Position := -1;
I := 0;
repeat
   Inc (I);
until (I = Size) or (Table[I] = Key);
if  Table[I] = Key  then  Position := I;
```

תרגיל 10.3

```
program SearchNumber;
var
   I ,L, H, Position:      Integer;
   Counter1, Counter2:  Integer;
   Table:                    array [1..9]  of  Integr;

begin
   Table[1] := 1;
   Table[2] := 6;
   Table[3] := 9;
   Table[4] := 11;
   Table[5] := 17;
   Table[6] := 18;
   Table[7] := 22;
   Table[8] := 24;
   Table[9] := 26;

   Counter1 := 0;
   Counter2 := 0;
   Position  := -1;
   for  I := 1  to  9  do
      begin
            Inc  (Counter1);
            if  Table[I]  =  22  then
               Position := I;
      end;

   L := 1;
   H := 9;

   repeat
      Inc (Counter2);
      I := (L + H) div 2;
      if  Table[I] < Key  then  L := I;
      if  Table[I] > Key  then  L := H;
   until (Table[I] = Key) or (L = H);

   Writeln ('Counter1 = ', Counter1);
   Writeln ('Counter2 = ', Counter2);
end.
```

תרגיל 10.4

```
program ListStudents;
type
   Student  =  record
      Name:        string[20];
      Marks:        Real;
   end;

var
   I:              Integer;
   Table:         array [1..20]  of  Student;

begin
   for  I := 1  to  20  do
      if  Table[I].Marks > 80.0  then
            begin
               Writeln  ('Name: ', Table[I].Name);
               Writeln  ('Marks average: ', Table[I].Marks);
            end;
end.
```

פרק 11

תרגיל 11.1

```
program Sort;
const
   N = 5;

var
   A, B, Temp:    Integer;
   Table:            array [1..N]  of  Integer;

begin
   for  A := 1  to  N - 1  do
      for  B := A + 1  to  N  do
            if  Table[B] < Table[A]  then
               begin
                       Temp := Table[A];
                       Table[A] := Table[B];
                       Table[B] := Table[A];
               end;
end.
```

תרגיל 11.2

```
program Merging;
var
   Pointer1:     Integer;
   Pointer2:     Integer;
   Pointer3:     Integer;
   Table1:       array [1..10]  of  Integer;
   Table2:       array [1..10]  of  Integer;
   Table3:       array [1..10]  of  Integer;

begin
   Pointer1 := 1;
   Pointer2 := 10;
   Pointer3 := 1;
   repeat
      if  Table1[Pointer1] <= Table2[Pointer2]  then
            begin
               Table3[Pointer3] := Table1[Pointer1];
               Pointer1 := Pointer1 + 1;
            end
      else
            begin
               Table3[Pointer3] := Tablel2[Pointer2];
               Pointer2 := Pointer2 - 1;
            end;
      pointer3 := pointer3 + 1;
   until  (Pointer1 > 10)  or  (Pointer2 < 1);

   if  Pointer1 <= 10  then
      while  Pointer1 <= 10  do
            begin
               Table3[Pointer3] := Table1[Pointer1];
               Pointer1 := Pointer1 + 1;
               Pointer3 := Pointer3 + 1;
            end
   else
      while  Pointer2 >= 1  do
            begin
               Table3[Pointer3] := Table2[Pointer2];
               Pointer2 := Pointer2 - 1;
               Pointer3 := Pointer3 + 1;
            end
end.
```

תרגיל 11.3

```
program Merging;
var
   A, B, Temp:          Integer;
   Pointer1:            Integer;
   Pointer2:            Integer;
   Switch:      Boolean;
   Table1:      array [1..10]  of  Integer;
   Table2:              array [1..10]  of  Integer;
   Table3:              array [1..10]  of  Integer;

begin
  { העתקת שני המערכים לתוך מערך שלישי }
   for  A := 1  to  10  do
      begin
            Table3[A] := Table1[A];
            Table3[A + 10] := Table2[A];
      end;

   {  מיון נתונים ע"פ שיטת מיון בועות }
   A := 1;
   Switch := True;
   while  (A < 10)  and (Switch = True)  do
      begin
            Switch := False;
            for  B :=1  to  20 - A  do
               if Table3[B + 1] < Table3[B]  then
                       begin
                     Temp := Table3[B + 1];
                     Table3[B + 1] := Table3[B];
                     Table3[B] := Temp;
                     Switch := True;
                  end;
               A := A + 1;
      end;
end.
```

תרגיל 11.4

```
program MergeAndFilter;
var
   Pointer1:     Integer;
   Pointer2:     Integer;
   Pointer3:     Integer;
```

```
    Table1:        array [1..6]  of  Real;
    Table2:        array [1..6]  of  Real;
    Table3:        array [1..12]  of  Real;

begin
    Pointer1 := 1;
    Pointer2 := 1;
    Pointer3 := 1;

    repeat
        while (Pointer1 < 6) and (Table1[Pointer1] < 0)    do
                Pointer1 := Pointer1 + 1;
        while (Pointer2 < 6) and (Table2[Pointer2] < 0)    do
                Pointer2 := Pointer2 + 1;
        if  (Pointer1 <= 6)  and  (Pointer2 <= 6)  then
                begin
                    if  Table1[Pointer1] <= Table2[Pointer2] then
                        begin
                            Table3[Pointer3] := Table1[Pointer1];
                            Pointer1 := Pointer1 + 1;
                        end
                    else
                        begin
                            Table3[Pointer3] := Tablel2[Pointer2];
                            Pointer2 := Pointer2 + 1;
                        end;
                end;
    until  (Pointer1 > 6)  or  (Pointer2 < 6);

    if  Pointer1 <= 6  then
        while  Pointer1 <= 6  do
                begin
                    Table3[Pointer3] := Table1[Pointer1];
                    Pointer1 := Pointer1 + 1;
                    Pointer3 := Pointer3 + 1;
                end
    else
        while  Pointer2 <= 6  do
                begin
                    Table3[Pointer3] := Table2[Pointer2];
                    Pointer2 := Pointer2 + 1;
                    Pointer3 := Pointer3 + 1;
                end
end.
```

תרגיל 11.5

```
program AddName;
var
   Pointer1:     Integer;
   Pointer2:     Integer;
   St:           string;
   Table1:       array [1..10]  of  string;
   Table2:       array [1..11]  of  string;

begin
   Writeln ('Type in a name:');
   Readln (St);
   Pointer1 := 1;
   for  Pointer2 := 1  to  11  do
      if  Table1[Pointer1] < St  then
            begin
               Table2[Pointer2] := Table1[Pointer1];
               Pointer1 := Pointer1 + 1;
            end
      else
            Table2[Pointer2] := St;
end.
```

תרגיל 11.6

```
program CompareSort;
var
   A, B, Temp: Integer;
   Counter1:   Integer;
   Counter2:   Integer;
   Switch:     Boolean;
   Table1:     array [1..5]  of  Integer;
   Table2:     array [1..5]  of  Integer;

begin
   { תיכול המערך הראשון, ושכפולו לשני }
   Table1[1] := 14;
   Table1[2] := 93;
   Table1[3] := 44;
   Table1[4] := 7;
   Table1[5] := 16;
```

```
    for  A := 1  to  5  do  Table2[A] := Table1[A];
    {  אתחול מונים }
       Counter1 := 0;
       Counter2 := 0;
       {  מיון נתונים ע"פ שיטת מיון שכנים  }
       for  A := 1  to  5  do
             for  B := 1  to  5  do
                begin
                   Counter1 := Counter1  +  1;
                   if  Table[B] < Table[A]  then
                      begin
                         Temp := Table[A];
                         Table[A] := Table[B];
                         Table[B] := Temp;
                      end;
                end;
    {  מיון נתונים ע"פ שיטת מיון בועות }
    A := 1;
    Switch := True;
    while  (A < 5)  and  (Switch = True)  do
       begin
             Counter2 := Counter2 + 1;
             Switch := False;
             for  B :=1  to  5 - A  do
                if   Table3[B + 1] < Table3[B]  then
                   begin
                      Temp := Table3[B + 1];
                      Table3[B + 1] := Table3[B];
                      Table3[B] := Temp;
                      Switch := True;
                   end;
             A := A + 1;
       end;

    {  הדפסת תוצאות }
    Writeln ('Counter  of  first  sort: ', Counter1);
    Writeln ('Counter  of  second  sort: ', Counter2);
end.
```

פרק 12

תרגיל 12.1

```
program ListFile;
var
   F:          Text;
   St:         string;

begin
   Assign (F, 'C:\AUTOEXEC.BAT');
   Reset (F);
   while  Eof (F) = False  do
      begin
            Readln (F, St);
            Writeln (St);
      end;
   Close (F);
end.
```

תרגיל 12.2

```
program WriteNumbers;
var
   F:            Text;
   I:         Integer;
   Table:     array [1..6]  of  Integer;

begin
   for  I := 1  to  6  do
      begin
            Writeln  ('Enter a value ', I);
            Readln  (Table[I]);
      end;
   Assign (F, 'NUMBERS.DAT');
   Rewrite (F);
   for  I := 1  to  6  do
      Writeln (F, Table[I]);
      Close (F);
end.
```

תרגיל 12.3

```
program ReadNumbers;
var
    F:              file of Integer;
    I,  A:      Integer;
    Table:          array [1..100]  of  Integer;

begin
    Assign (F, 'LIST.DAT');
    Reset (F);
    I := 0;
    while  Eof (F) = False  do
        begin
                Inc (I);
                Read (F, Table[I]);
                Read (F, A);
        end;
    Close (F);
    Writeln ('Total numbers in table: ', I);
end.
```

תרגיל 12.4

```
program WriteStrings;
var
    F:            Text;
    St:           string;

begin
    Assign (F, 'LINE.TXT');
    Rewrite (F);
    repeat
        Readln (St);
        Writeln (F, St);
    until  St = ' ';
    Close (F);
end.
```

תרגיל 12.5

```
program PersonalFile;
type
    Data = record
        Name:           string;
        Age:            Integer;
        Vetek:          Integer;
    end;

var
    F:          file of Data;
    P:          Data;
```

תרגיל 12.6

```
program PhoneBook;
type
    Data = record
        FirstName:      string[15];
        LastName:       string[20];
        PhoneNumber:    string[10];
    end;

var
    A:          Data;
    F:          file of Data;

begin
    Assign (F, 'PHONE.DAT');
    Rewrite (F);
    repeat
        Write ('First name: ');
        Readln (A.FirstName);
        if  A.FirstName <> '000'  then
                begin
                    Write ('Last name: ');
                    Readln (A.LastName);
                    Write ('Phone number: ');
                    Readln (A.Phone);
                    Write (F, A);
                end;
    until  A.FirstName = '000';
    Close (F);
end.
```

תרגיל 12.7

```
program PhoneBook;
type
    Data = record
        FirstName:      string[15];
        LastName:       string[20];
        PhoneNumber:    string[10];
    end;

var
    A:            Data;
    F:            file of Data;

begin
    Assign (F, 'PHONE.DAT');
    Reset (F);
    while  Eof (F) = False  do
        begin
                Read (F, A);
                Write (A.FirstName:16, A.LastName:21, A.Phone:11);
        end;
    Close (F);
end.
```

תרגיל 12.8

```
program CopyFile;
var
    F1, F2:            Text;
    St:                string;
    Name1,  Name2:     string;

begin
    Write ('Enter first file name: ');
    Readln (Name1);
    Write ('Enter second file name: ');
    Readln (Name2);
    Assign (F1, Name1);
    Reset (F1);
    Assign (F2, Name2);
    Rewrite (F2);
```

```
    while Eof (F1) = False  do
        begin
                Readln (F1, St);
                Writeln (F2, St);
        end;

    Close (F1);
    Close (F2);
end.
```

תרגיל 12.9

```
program CopyData;
var
    F1, F2, F3:    file of Integer;
    I:             Integer;

begin
    Assign (F1, 'FILE1.DAT');
    Assign (F2, 'FILE2.DAT');
    Assign (F3, 'FILE3.DAT');
    Reset (F1);
    Reset (F2);
    Rewrite (F3);

    while  Eof (F1) = False  do
        begin
                Read  (F1,  I);
                Write  (F3,  I);
                Read  (F2,  I);
                Write  (F3,  I);
        end;

    Close (F1);
    Close (F2);
    Close (F3);
end.
```

אינדקס